UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
TELEMÁTICA

Exposición de APIs de red de operador, a desarrolladores, basada en orientación a objetos

BlueVia 1.6 ***.NET*** SDK

AUTOR: Isidro Gómez Crisóstomo
TUTOR: Pablo Basanta Val
DIRECTOR Enrique Vélez Tarilonte

Febrero de 2013

Memoria del proyecto de fin de carrera
realizado en *Telefónica I+D*

“El cielo sobre el puerto tenía el color de la pantalla de un televisor sintonizado en un canal muerto.”

William Gibson, Neuromante.

AGRADECIMIENTOS:

*Me gustaría agradecer a mis **padres** y a mi **familia** el apoyo e infinita paciencia que me han brindado durante todo este tiempo.*

*También a los guías de este proyecto: **Kike** y **Pablo**, por mostrarme el camino; y a **Robert**, que me lo cedió y orientó, nada más estrenarlo.*

*A todos los que me han ayudado en el viaje que parece concluir aquí -profesores y compañeros-; en particular a aquellos cuya compañía ha sido más intensa: **Ari**, **Elo**, **Ángel**, **Jesús**, **Luís**, **David**; y muy especialmente, a mi excelente binomio: **Faty**, por completarme y animarme en la carrera; y a mi confidente: **Bea**, por inspirarme y sorprenderme siempre.*

Muchas, muchas gracias:)

RESUMEN

Gracias al auge de los servicios web, muchas empresas intentan abrir nuevos nichos de mercado con ellos. ***Telefónica*** ha desarrollado la plataforma de servicios ***Bluevia***, ofreciendo funcionalidad relacionada con las redes de telefonía móvil a través de una API web, con la intención de que pueda ser integrada de forma masiva en las nuevas generaciones de aplicaciones software. Para ello, ofrece de forma gratuita varios kits de desarrollo opensource en diferentes lenguajes de programación, que facilitan el acceso a estos servicios. El presente proyecto fin de carrera, realizado en ***Telefónica I+D***, consiste en el desarrollo de uno de estos kits, compuesto por unas librerías creadas en el lenguaje de programación ***C#*** que permita acceso a los servicios de la versión 1.6 de la plataforma, y la documentación que facilita su uso.

En esta memoria se realiza una introducción al entorno en el que tiene que desenvolverse el SDK desarrollado en este proyecto. Tras ello, se abordan los aspectos relativos al análisis de los requisitos demandados inicialmente para el producto, y el diseño en consecuencia de una estructura general que cumpla con ellos de forma flexible. Para finalizar, se describe la implementación de la solución diseñada en el ámbito específico del lenguaje ***C#*** sobre el ***.NET framework 4*** de ***Microsoft***; junto con el desarrollo de la documentación y aplicación de demostraciones relacionadas; además de varias muestras de la validación de las librerías creadas.

ÍNDICE DE CONTENIDO

ÍNDICE DE CONTENIDO 1
ÍNDICE DE FIGURAS 4
Estilo particular de esta memoria. 13
1 INTRODUCCIÓN 14
1.1 OBJETIVOS 14
1.2 ESTRUCTURA DE LA MEMORIA 15
2 ESTADO DEL ARTE 16
2.1 *BLUEVIA* 17
2.1.1 Actores en *Bluevia* 18
2.1.2 Puntos de acceso de la plataforma 19
2.1.3 Aplicaciones e identificadores 21
2.1.4 Modos de uso operativo 23
2.1.5 APIs de *Bluevia* 24
2.1.6 Los SDK de acceso a los servicios de *Bluevia* 26
2.1.7 Limitaciones de las versiones anteriores del SDK para *.NET* 27
2.2 APIs WEB 30
2.2.1 HTTP 31
2.2.2 HHTP para *Bluevia* 32
2.3 *OAUTH* 35
2.3.1 *OAuth* para *Bluevia* 37
2.4 *.NET FRAMEWORK* 4 39
2.4.1 Algunas novedades del Framework 4 39
2.4.2 Visual Studio2010 41
3 DISEÑO DE LA VERSION 1.6 DEL SDK DE ACCESO A BLUEVIA PARA *.NET* 42
3.1 REQUISITOS PARA EL NUEVO SDK 43
3.1.1 Requisitos de alto nivel, heredados de las versiones previas del SDK 43
3.1.2 Requisitos adicionales de alto nivel, para la nueva versión del SDK 43
3.2 ANÁLISIS DE REQUISITOS 45
3.2.1 Análisis de los requisitos de acceso a los servicios 45
3.2.2 Análisis del requisito sobre servicios privados 51
3.2.3 Análisis del requisito de armonización entre lenguajes 52
3.2.4 Análisis del requisito de documentación y demos 52
3.2.5 Análisis de la experiencia con las versiones anteriores del SDK. 52
3.3 DISEÑO DE MEDIO NIVEL 54
3.3.1 Nivel de acceso a la red 56

3.3.2 Nivel de inteligencia común 60
3.3.3 Nivel de inteligencia de API 64
3.4 DISEÑO DE BAJO NIVEL 71
3.4.1 Herencia en los módulos diseñados 71
3.4.2 Agrupando la funcionalidad 74
3.5 RESUMEN DEL DISEÑO 76
4 IMPLEMENTACIÓN Y VALIDACIÓN DEL PROYECTO 78
4.1 SOFTWARE UTILIZADO PARA LA REALIZACIÓN DEL PAQUETE ENTREGABLE 79
4.1.1 Sistema operativo 79
4.1.2 Software para el desarrollo de la librería 79
4.1.3 Software para el desarrollo de la documentación 80
4.1.4 Software para el control de versiones y entregas 80
4.1.5 Otros 81
4.2 IMPLEMENTACIÓN DE LAS LIBRERÍAS 82
4.2.1 Sistema de paquetes 82
4.2.2 Descripción de clases 83
4.2.3 Unificación de la interfaz expuesta al usuario 134
4.2.4 Problemas encontrados 134
4.3 IMPLEMENTACIÓN DE LAS DEMOS 136
4.3.1 Descripción de clases de los ejemplos públicos 136
4.3.2 Diferencias entre los ejemplos públicos y privados 142
4.4 VALIDACIÓN DE LAS LIBRERÍAS 143
4.4.1 Validación de *Advertising* 144
4.4.2 Validación de *Directory* 145
4.4.3 Validación de *Location* 150
4.4.4 Validación de MMS 151
4.4.5 Validación de *OAuth* 158
4.4.6 Validación de *Payment* 160
4.4.7 Validación de SMS 163
4.4.8 Validación de control y captura de Errores 167
4.5 GENERACIÓN DE LA DOCUMENTACIÓN 168
4.5.1 Documentación del código y referencias de clase 168
4.5.2 Guías de uso de las API 169
4.5.3 Guías de uso del SDK 170
4.5.4 Documentación del paquete de acceso privado 171
5 HISTORIA DEL PROYECTO 172
6 CONCLUSIONES 174
6.1 Objetivos logrados 174

6.2 Líneas futuras de trabajo 174
7 ANEXOS 176
7.1 PRESUPUESTO 177
7.2 RESUMEN DE LAS ESPECIFICACIONES PARA EL ACCESO A LOS SERVICIOS DE BLUEVIA 178
7.3 DESCRIPCIÓN DE LOS SERVICIOS OFRECIDOS POR *BLUEVIA 1.6* 184
7.3.1 OAuth 184
7.3.2 Advertising 184
7.3.3 Directory 185
7.3.4 Location 185
7.3.5 MMS y SMS 185
7.3.6 Payment 186
7.3.7 Certificación 186
7.3.8 Compartición de beneficios 186
7.4 INSTALACIÓN DEL ENTORNO DE DESARROLLO 188
7.4.1 *.NET* FRAMEWORK 4 188
7.4.2 VISUAL STUDIO 2010 188
7.5 PROXIES Y MONITORIZACIÓN DE PETICIONES 190
7.5.1 WhireShark 190
7.5.2 Fiddler2 191
7.6 ESTRUCTURA DEL REPOSITORIO 195
7.7 CUENTA DE DESARROLLADOR DE *BLUEVIA*, CREACIÓN DE CREDENCIALES 196
7.7.1 Manejo del panel de desarrollador 196
7.7.2 Creación de credenciales de aplicación 197
7.8 CREACIÓN DE CREDENCIALES PARA EL USUARIO FINAL 201
7.9 DOCUMENTACIÓN CON DOXYGEN 203
7.9.1 Un ejemplo de la documentación generada 203
7.10 GLOSARIO DE TÉRMINOS 207
7.11 REFERENCIAS DE LA MEMORIA 209
7.12 BIBLIOGRAFÍA 212
7.12.1 Bibliografía del SDK 212

ÍNDICE DE FIGURAS

Figura 2-1 Imagen esquemática de los servicios de *Bluevia* 17

Figura 2-2 Imagen esquemática de las relaciones entre los actores de *Bluevia* 19

Figura 2-3 Fragmento de la página principal del portal del *Bluevia: Portal* 19

Figura 2-4 Portada del portal de autorización de *Bluevia* para España: *Connect* 20

Figura 2-5 Esquema de credenciales de *Bluevia* para los servicios de acceso público 22

Figura 2-6 Esquema de credenciales de *Bluevia* para los servicios de acceso privado 22

Figura 2-7 Imagen de cabecera HTTP *Authorization:OAuth 2legged*, sin credenciales de usuario, opcionalmente puede aparecer el campo vacío 23

Figura 2-8 Imagen de cabecera HTTP *Authorization:OAuth 3legged* 23

Figura 2-9 Tabla de diferencias entre los modos de uso de *Bluevia* 24

Figura 2-10 Tabla de disponibilidad de servicios de *Bluevia* por país. Fuente: *Bluevia* 25

Figura 2-11 Captura de una petición para la descarga de un anuncio, en la que pueden apreciarse los parámetros de búsqueda 26

Figura 2-12 Captura de una respuesta, con datos de un anuncio acorde a los requisitos demandados previamente. 26

Figura 2-13 Código comparativo de la creación de un cliente y la instanciación de uno de sus métodos, entre los SDKs de *Ruby* y *Java,* para su versión 1.5 28

Figura 2-14 Código de la creación de un cliente y la instanciación de uno de sus métodos, para la versión 1.5 del SDK de *.NET* 28

Figura 2-15 Estructura general del SDK previo a la versión 1.6 29

Figura 2-16 Imagen de las partes de una transacción HTTP, y leyenda 31

Figura 2-17 Imagen de petición HTTP con cuerpo en formato x-www-form-urlencoded 33

Figura 2-18 Imagen de respuesta HTTP con cuerpo en formato XML, para comunicación SOAP 33

Figura 2-19 Imagen de petición HTTP con cuerpo en formato XML, para comunicación XML-RPC 34

Figura 2-20 Imagen de respuesta HTTP con cuerpo en formato multipart, con 3 archivos contenidos: un XML, un texto, y una imagen 34

Figura 2-21 Imagen esquemática de las funciones ofrecidas por OAuth 35

Figura 2-22 Proceso *OAuth* de autorización 37

Figura 2-23 Esquema de la estructura .NET 39

Figura 2-24 Fragmentos de código que ilustran la nueva funcionalidad del *framework 4* para *C#*, de los parámetros opcionales. 40

Figura 2-25 Fragmentos de código que ilustran la nueva funcionalidad del *framework 4* para *C#*, de los parámetros nombrados. 40

Figura 3-1 Tabla de los requisitos generales para el proyecto. 45

Figura 3-2 Esquema de los pasos a realizar para invocar un servicio de *Bluevia* 47

Figura 3-3 Ejemplo de petición al API de *Advertising* 48

Figura 3-4 Ejemplo de la respuesta del API de *Advertising* 49

Figura 3-5 Distribución del trabajo para el usuario y el SDK, en los pasos a realizar para invocar un servicio de *Bluevia* 50

Figura 3-6 Tabla de las directrices generales de diseño para el proyecto. 54

Figura 3-7 Estructura general del SDK versión 1.6 55

Figura 3-8 Detalle del Nivel de paso de mensajes 57

Figura 3-9 Flujo de la funcionalidad de los métodos CRUD en el módulo HTTPS 58

Figura 3-10 Flujo de montaje, envío y recuperación de una petición http/*Bluevia* 58

Figura 3-11 Representación gráfica de una respuesta genérica de *Bluevia.* 59

Figura 3-12 Detalle del nivel de inteligencia común 61

Figura 3-13 Flujo de la funcionalidad de los métodos CRUD en el cliente básico 62

Figura 3-14 Descomposición del módulo de “Serializadores” 63

Figura 3-15 Descomposición del módulo de “Parseadores” 64

Figura 3-16 Detalle del nivel de inteligencia de API 65

Figura 3-17 Detalle de la estructura del cliente común de un API. 66

Figura 3-18 Flujo de la invocación de un servicio desde la perspectiva del nivel de inteligencia de API 67

Figura 3-19 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *OAuth*, y los servicios invocados de la plataforma *Bluevia*. 68

Figura 3-20 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Advertising*, y los servicios invocados de la plataforma *Bluevia*. 68

Figura 3-21 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Directory*, y los servicios invocados de la plataforma *Bluevia*. 69

Figura 3-22 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Location*, y los servicios invocados de la plataforma *Bluevia* 69

Figura 3-23 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de SMS, y los servicios invocados de la plataforma *Bluevia*. 69

Figura 3-24 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de MMS, y los servicios invocados de la plataforma *Bluevia*..... 70

Figura 3-25 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Payment*, y los servicios invocados de la plataforma *Bluevia*. 70

Figura 3-26 Diagrama de herencia entre clientes de *Bluevia*..... 71

Figura 3-27 Diagrama de herencia para los API de Mensajería 72

Figura 3-28 Diagrama de herencia para los API de Payment y OAuth. 72

Figura 3-29 Diagrama de herencia entre conectores de *Bluevia* 73

Figura 3-30 Esquema de integración de la funcionalidad del proyecto. 74

Figura 3-31 Esquema de inicialización encadenada. 75

Figura 4-1 Diagrama de conectores e interfaces del nivel de acceso a la red. 84

Figura 4-2 Diagrama de clases auxiliares del nivel de acceso a la red 88

Figura 4-3 Diagrama de clases de constantes del nivel de acceso a la red 92

Figura 4-4 Diagrama de clases devueltas por el nivel de acceso a la red..... 93

Figura 4-5 Diagrama de clientes base del nivel de inteligencia común 95

Figura 4-6 Diagrama de los serializadores..... 98

Figura 4-7 Fragmento de código que permite el *casting* de un objeto recibido sin tipo, al tipo lista de adjuntos..... 98

Figura 4-8 Fragmento de código que permite el *casting* de un objeto recibido sin tipo, al tipo mensaje *Multipart* 99

Figura 4-9 Fragmento de código que permite preparar al serializador para aplicar el formato XML adecuado a un tipo T recibido 100

Figura 4-10 Fragmento de código del FormUrlSerlializer para unificar parámetros en una sola cadena 101

Figura 4-11 Diagrama de los “Parseadores”..... 101

Figura 4-12 Fragmento de código de los “Parseadores” que transforma el cuerpo del mensaje en un texto procesable 101

Figura 4-13 Fragmento de código para usar las librerías de “deserialización” de *framework* 103

Figura 4-14 Diagrama de clientes de *Oauth* 106

Figura 4-15 Diagrama del simplificador de *Oauth*..... 107

Figura 4-16 Diagrama de las clases contenedoras y constantes *de Oauth* 108

Figura 4-17 Diagrama de clientes de *Advertising* 109

Figura 4-18 Diagrama de clases auxiliares de *Advertising* 110

Figura 4-19 Diagrama de las clases contenedoras y constantes *de Advertising* 111

Figura 4-20 Diagrama de clientes de *Directory* 113

Figura 4-21 Diagrama de clases auxiliares de *Directory* 114

Figura 4-22 Diagrama de objetos contenedores de *Directory* 115

Figura 4-23 Diagrama de constantes y enumerados de *Directory* 116

Figura 4-24 Diagrama de clientes de *Location* 117

Figura 4-25 Diagrama del simplificador de *Location* 118

Figura 4-26 Diagrama de las clases contenedoras y constantes *de Location* 118

Figura 4-27 Diagrama de clientes de Mensajería SMS 119

Figura 4-28 Diagrama de clientes de mensajería MMS 121

Figura 4-29 Fragmento de código con el que se montan los objetos *MIMEContent* en el método *GetAttachment* 124

Figura 4-30 Diagrama de clases auxiliares de SMS 124

Figura 4-31 Diagrama de clases auxiliares de MMS 125

Figura 4-32 Diagrama de clases contenedoras y constantes de mensajería 126

Figura 4-33 Diagrama de clases contenedoras y constantes de SMS 127

Figura 4-34 Diagrama de clases contenedoras de MMS 128

Figura 4-35 Diagrama de constantes y enumerados de MMS 130

Figura 4-36 Diagramas de clientes de *Payment* 131

Figura 4-37 Diagrama del simplificador de *Payment* 132

Figura 4-38 Diagrama de los objetos contenedores y constantes de *Payment* 133

Figura 4-39 Código comparativo de la creación de un cliente, la instanciación de uno de sus métodos, y la recuperación de la información obtenida; entre los SDKs de *PHP* y *C#* 134

Figura 4-40 Diagrama de clases principales del paquete de ejemplos 136

Figura 4-41 Vista del menú principal de la aplicación de demos Example_Launcher 137

Figura 4-42 Vista de la pantalla de ejecución del ejemplo de *Directory* 138

Figura 4-43 Código de invocación de *Example_Advertising* 144

Figura 4-44 Captura de la consola al ejecutar *Example_Advertising* 144

Figura 4-45 Captura de las trazas de red de la ejecución de *Example_Advertising* 144

Figura 4-46 Código de invocación de *Example_Directory* 145
Figura 4-47 Captura de la consola al ejecutar *Example_Directory* 145
Figura 4-48 Captura de las trazas de red al ejecutar *Example_Directory* 145
Figura 4-49 Código de invocación en *Example_Directory_Access* 146
Figura 4-50 Captura de la consola al ejecutar *Example_Directory_Access* 146
Figura 4-51 Captura de las trazas de red al ejecutar *Example_Directory_Access* 146
Figura 4-52 Código de invocación en *Example_Directory_Personal* 147
Figura 4-53 Captura de la consola al ejecutar *Example_Directory_Personal* 147
Figura 4-54 Captura de las trazas de red al ejecutar *Example_Directory_Personal* 147
Figura 4-55 Código de invocación en *Example_Directory_Profile* 148
Figura 4-56 Captura de la consola al ejecutar *Example_Directory_Profile* 148
Figura 4-57 Captura de las trazas de red al ejecutar *Example_Directory_Profile* 148
Figura 4-58 Código de invocación en *Example_Directory_Terminal* 149
Figura 4-59 Captura de la consola al ejecutar *Example_Directory_Terminal* 149
Figura 4-60 Captura de las trazas de red al ejecutar *Example_Directory_Terminal* 149
Figura 4-61 Código de invocación en *Example_Location* 150
Figura 4-62 Captura de la consola al ejecutar *Example_Location* 150
Figura 4-63 Captura de las trazas de red al ejecutar *Example_Location* 150
Figura 4-64 Código del envío de un MMS con un cliente MMS_MT, de *Example_MMS_MT* 151
Figura 4-65 Código de recuperación del estado de envío del MMS, de *Example_MMS_MT* 151
Figura 4-66 Captura de la consola al ejecutar *Example_MMS_MT* 151
Figura 4-67 Captura de las trazas de red al enviar un MMS 152
Figura 4-68 Captura de las trazas de red al recuperar el estado del envío de un MMS 153
Figura 4-69 Código de envío de un MMS a un buzón, de *Example_MMS_MO* 153
Figura 4-70 Código de recuperación de la lista de MMS del buzón, de *Example_MMS_MO* 154
Figura 4-71 Código de recuperación del primer MMS del buzón, de *Example_MMS_MO* . 154
Figura 4-72 Captura de la consola al ejecutar *Example_MMS_MO* 154
Figura 4-73 Captura de las trazas de red al recuperar la lista de mensajes existentes en el buzón 155

Figura 4-74 Captura de las trazas de red de la descarga de un mensaje completo, desde un buzón de MMS 155

Figura 4-75 Código de *Example_Notifications* modificado para dar de alta el servicio de notificaciones de MMS..... 156

Figura 4-76 Código de *Example_Notifications* modificado para dar de baja el servicio de notificaciones de MMS..... 156

Figura 4-77 Captura de la consola al ejecutar *Example_MMS_Notifications* 156

Figura 4-78 Captura de las trazas de red de la subscripción a las notificaciones de MMS .. 157

Figura 4-79 Captura de las trazas de red de la baja para las notificaciones de MMS..... 157

Figura 4-80 Código para la recuperación de *Tokens* temporales, en *Example_OAuth* 158

Figura 4-81 Código para la recuperación de *Tokens* definitivos, en *Example_OAuth* 158

Figura 4-82 Captura de la consola al ejecutar *Example_OAuth* 159

Figura 4-83 Captura de las trazas de red de la petición de *Tokens* temporales 159

Figura 4-84 Captura de las trazas de red de la petición de *Tokens* definitivos 159

Figura 4-85 Código para autorizar la transacción de *Example_Payment*..... 160

Figura 4-86 Captura de las trazas de red de la petición de *Tokens* temporales para autorizar una transacción 160

Figura 4-87 Captura del fallo en la autorización de la transacción en *Connect* 161

Figura 4-88 Código para ejecutar la transacción de *Example_Payment* 161

Figura 4-89 Captura del tráfico generado durante una petición de pago de transacción ... 162

Figura 4-90 Código para recuperar el estado de la transacción de *Example_Payment*..... 162

Figura 4-91 Código para cancelar la autorización de la transacción de *Example_Payment* 162

Figura 4-92 Código del envío de un SMS con un cliente SMS_MT, de *Example_SMS_MT* .. 163

Figura 4-93 Código de recuperación del estado de envío del SMS, de *Example_SMS_MT*. 163

Figura 4-94 Captura de la consola al ejecutar *Example_SMS_MT* 163

Figura 4-95 Captura de las trazas de red al enviar un SMS 164

Figura 4-96 Captura de las trazas de red al recuperar el estado del envío de un SMS..... 164

Figura 4-97 Código del envío de un SMS con un cliente SMS_MT, de *Example_SMS_MO* . 165

Figura 4-98 Código dela recuperación de todos los SMS de un buzón, de *Example_SMS_MO*. 165

Figura 4-99 Captura de la consola al ejecutar *Example_SMS_MO* 165

Figura 4-100 Captura de las trazas de red al recuperar los mensajes del buzón..... 166

Figura 4-101 Código de valores erróneos..... 167

Figura 4-102 Captura de pantalla de un error controlado por la librería, durante la ejecución de la aplicación de demos..... 167

Figura 4-103 Cabecera informativa de *Bluevia* insertada como comentario en todas las clases no autogeneradas de la librería..... 168

Figura 4-104 Ejemplo de comentarios justificando sentencias de código 169

Figura 4-105 Ejemplo de comentario enlazando a un recurso online 169

Figura 7-1 Tabla de requisitos exigidos por el servicio de OAuth. 179

Figura 7-2 Tabla de requisitos exigidos por el servicio de *Advertising*..... 179

Figura 7-3 Tabla de requisitos exigidos por el servicio de *Directory*. 180

Figura 7-4 Tabla de requisitos exigidos por el servicio de *Location*. 180

Figura 7-5 Tabla de requisitos exigidos por el servicio de SMS. 181

Figura 7-6 Tabla de requisitos exigidos por el servicio de MMS. 182

Figura 7-7 Tabla de requisitos exigidos por el servicio de *Payment*. 183

Figura 7-8 Tabla de compartición de beneficios por país 187

Figura 7-9 Pantalla de la selección de opciones en la instalación de *Visual C# 2010 Express* 188

Figura 7-10 Pantalla de descarga de los archivos de instalación de *Visual C# 2010 Express*189

Figura 7-11 Pantalla del proceso de instalación de *Visual C# 2010 Express* 189

Figura 7-12 Flujo del intercambio de mensajes para una comunicación SSL de dos vías Fuente: *CISCO* [15] 190

Figura 7-13 Imagen de la apertura de la configuración de opciones en Fiddler2..... 191

Figura 7-14 Imagen de la configuración para “desencriptar” tráfico HTTPS en Fiddler2 192

Figura 7-15 Imagen del inicio de la captura de tráfico en Fiddler2..... 192

Figura 7-16 Imagen de la captura del tráfico de red generado al ejecutar el ejemplo de *Advertising.* 193

Figura 7-17 Imagen del despliegue de la información relativa a la comunicación con el servicio de *Advertising* 193

Figura 7-18 Imagen de las vistas “*Raw”* para la visualización detallada de los mensajes de red 194

Figura 7-19 Vista principal del panel de desarrollador de *Bluevia*, y leyenda. 196

Figura 7-20 Imagen de la subsección “*My API Keys”*, con dos aplicaciones listadas. 197

Figura 7-21 Imagen de la apertura de la sección de creación de credenciales para aplicaciones. 198

Figura 7-22 Imagen de la sección básica del formulario de alta de aplicaciones en *Bluevia*, y leyenda. 198

Figura 7-23 Imagen de la sección de selección de API del formulario de alta de aplicaciones en *Bluevia*, y leyenda..... 199

Figura 7-24 Imagen de la información referente a una aplicación del desarrollador..... 200

Figura 7-25 Imagen de las condiciones de la autorización de una aplicación por parte del usuario final 201

Figura 7-26 Imagen de la pantalla con las condiciones aceptadas, y el código de verificación. 202

Figura 7-27 Imagen de la pantalla de las aplicaciones autorizadas por el usuario 202

Figura 7-28 Imagen de una de las pantallas de configuración del *Doxywizard* 203

Figura 7-29 Ejemplo de las referencias de clase: BV_Connector 204

Figura 7-30 Ejemplo de las guías de uso de API: *Advertising.* 205

Figura 7-31 Ejemplo de los Manuales de uso del SDK: Introducción 206

Estilo particular de esta memoria.

En esta memoria se han seguido determinadas reglas de estilo que se justifican a continuación.

- **Figuras:**

  -Las notas carecerán de un título descriptivo, a diferencia del resto de las figuras insertadas en el texto.

  -Las notas y los fragmentos de código fuente aparecerán con el ancho correspondiente al párrafo en el que se encuentren. El resto de figuras se centrarán con respecto a la página completa.

- **Términos:**

  -A pesar de que en Castellano, el plural de unas siglas es invariable; para evitar ambigüedades en términos como API, se usará la costumbre inglesa de construir el plural añadiendo una "s", cuando no pueda extraerse el número por el contexto.

- **Estilos de fuente:**

  -El código fuente llevará el color por defecto que le imprime ***VisualStudio***.

  -MAYÚSCULAS: Serán escritas en esta capitalización las siglas, los títulos de primer y segundo orden, y los nombres de archivos y clases que así se hayan especificado.

  -*Cursiva*: Serán escritas con este estilo, las palabras inglesas y los métodos de las clases que aparezcan en los textos.

  -**Negrita**: Serán escritos con este estilo los títulos de los puntos, los elementos abstractos del diseño, y las notas

  -***Negrita y Cursiva***: Serán escritos con este estilo los nombres comerciales de empresas o productos que aparezcan en los textos; con la excepción de la palabra "*Bluevia"* que debido al número de apariciones podría llegar a entorpecer la lectura.

  -"Comillas": Irán entrecomilladas, las citas, los términos que quieran destacarse como objetivos de una descripción; y los términos provenientes del Inglés y adaptados al Castellano, que por la necesidad de significado no puedan traducirse adecuadamente.

  - -Comentarios-: Se usará la forma inglesa de comentario, delimitado entre guiones. Para enfatizar la idea de texto prescindible, serán escritos en gris.

  -(Paréntesis): Se escribirán entre paréntesis, piezas de información que completen la información a algo recién expresado, aunque en otro contexto.

  -Subrayado: Se subrayarán las ideas clave dentro de un texto.

1 INTRODUCCIÓN

Los SDK en diferentes lenguajes de programación, para el acceso a los API de la plataforma de servicios *Bluevia,* de ***Telefónica****;* han existido desde el lanzamiento de la versión 1.0 de la plataforma, y han ido evolucionando e incrementándose a la vez que esta.

Pero el aumento y cambio de los servicios y tecnologías implicadas en los API, provocan la necesidad de modificar radicalmente el núcleo de estos SDK; ello, sumado a la inexistencia de una estructura común entre los diferentes lenguajes, y teniendo en cuenta que son entregados como código abierto; hace patente la necesidad de un nuevo diseño y desarrollo de estos.

En este proyecto se acomete la tarea de crear la versión 1.6 del SDK para el entorno de ***.NET***, en el lenguaje de programación ***C#***; como parte del trabajo desarrollado ***Telefónica I+D*** durante las prácticas profesionales.

1.1 OBJETIVOS

El principal objetivo del este proyecto, es el de la realización de unas librerías en ***C#***, que faciliten el acceso a los servicios ofrecidos por *Bluevia* 1.6, para la versión 4 del *framework* ***.NET***. Con la intención de unificar la estructura y funcionamiento del código con el de los SDK del resto de lenguajes.

Para ello, deben de abordarse los siguientes objetivos:

- **Analizar los requisitos demandados por la dirección de la plataforma.**
- **Analizar las especificaciones de los servicios de la plataforma.**

  Estudio de las tecnologías implicadas en la comunicación con los servicios, y particularidades de los mismos.

- **Colaborar con los desarrolladores del resto de SDKs, para diseñar una estructura que permita el uso unificado de estos.**

  Recopilación de las particularidades de las versiones anteriores de los SDKs, para combinar las soluciones más adecuadas ya existentes en una estructura común, y sentar unas bases de un modo de desarrollo unificado.

- **Completar el diseño para adecuarlo al lenguaje *C#* e implementar la solución.**

  Aplicar las bases del desarrollo unificado, para diseñar la implementación de la estructura común de acuerdo a las posibilidades del lenguaje ***C#***.

Por otro lado, fuera de los objetivos formales, y dada la experiencia previa con las versiones anteriores de la librería, se pretende realizar el proyecto procurando evitar -o al menos facilitar- los aspectos que puedan dificultar los trabajos de mantenimiento futuros del SDK.

Nota: A pesar de que el objetivo de este proyecto es el de desarrollar un SDK de acceso a todos los servicios de *Bluevia 1.6,* debido a la naturaleza del negocio de los servicios privados, el ámbito de esta memoria se alejará bastante de estos; atendiendo casi en exclusiva a lo concerniente al acceso de los servicios públicos.

1.2 ESTRUCTURA DE LA MEMORIA

Además de portada e índices, la presente memoria está estructurada en 6 bloques compuestos de varios puntos:

1. **Introducción:** Comprende el presente bloque; en el que se pretende situar el proyecto, describir brevemente los objetivos que acomete, y describir la estructura de esta memoria.

2. **Estado del arte:** Se realiza una breve presentación de las tecnologías más importantes implicadas en el proyecto. Con puntos dedicados a la plataforma de servicios *Bluevia*, los APIs web, *OAuth*, y el *framework* 4 de ***.NET***.

3. **Diseño de la versión 1.6 del SDK de acceso a *Bluevia* para *.NET*:** Se plantean y analizan los requisitos del SDK, y tras ello se emprende el diseño de la solución a varios niveles.

4. **Implementación y validación del proyecto:** Este bloque abarca todo lo relativo al desarrollo de la solución diseñada. Se describe el entorno software utilizado, la implementación de las librerías y los ejemplos de demostración, la validación de las librerías, y la generación de la documentación del SDK.

5. **Historia del proyecto:** Breve descripción del proceso de desarrollo del SDK, llevado a cabo en ***Telefónica I+D***; y los aspectos de la plataforma que han cambiado desde la finalización del proyecto en la empresa.

6. **Conclusiones:** Se repasarán los objetivos logrados, y aquellas cuestiones mejorables; Así como las posibles ampliaciones o trabajos futuros que pueden desarrollarse a partir del SDK.

7. **Anexos:** En este bloque se desarrollan varios puntos de documentación referenciada a lo largo del resto del proyecto; además de contener el presupuesto del proyecto, una breve descripción de algunos de los cambios surgidos en Bluevia desde la entrega del SDK, el glosario de términos y la bibliografía.

2 ESTADO DEL ARTE

El contenido de este bloque consiste en una serie de puntos en los que se realiza una breve presentación de las tecnologías más importantes implicadas en el proyecto.

- El punto 2.1, presenta la plataforma de servicios *Bluevia,* de ***Telefónica***; describiendo los aspectos clave de su funcionamiento, esenciales para seguir el desarrollo de esta memoria.

- El punto 2.2, contiene una breve introducción sobre los APIs web, enfocada a cubrir los aspectos tecnológicos necesarios para comprender como se ofrecen por parte de *Bluevia*, los servicios a los que se accede en este proyecto

- El punto 2.2, contiene una descripción del funcionamiento de la tecnología de autorización *OAuth*, enfocada a completar los dos puntos anteriores en cuanto a los requisitos de los servicios de *Bluevia*.

- Finalmente en el punto 2.4, se hace una breve descripción de los aspectos más relevantes para el presente proyecto, del ***Microsoft .NET Framework 4***, y el IDE ***VisualStudio* 2010**.

2.1 *BLUEVIA*

Bluevia es una plataforma global, implementada por el grupo ***Telefónica*** desde finales del 2010, que ofrece servicios ligados a la telefonía móvil, a través de varios API web.

Está dirigida a todo tipo de desarrolladores, en especial a los pequeños estudios y desarrolladores independientes, a los que con servicios públicos, facilita la labor de integrar en sus aplicaciones las siguientes funcionalidades:

- Selección y recuperación de anuncios en formato imagen o texto: ***Advertising API***.
- Selección y recuperación de información asociada a un usuario de la red móvil: ***User Context API***.
- Recuperación de la localización por redes de telefonía, de un dispositivo móvil: ***Location API***.
- Envío y recepción de mensajería SMS y MMS: ***SMS y MMS API***.
- Pagos de bienes a través de la factura telefónica: ***Payment API***.

Bluevia mantiene una cartera potencial de más de doscientos millones de usuarios, ya que sus servicios están disponibles en todos los países en los que opera Telefónica. Y prácticamente no impone limitaciones de plataforma o lenguaje de desarrollo, ya que los API están expuestos como servicios web, y se accede a ellos mediante HTTPS; la plataforma ofrece la posibilidad de aprovechar un modelo de negocio interesante, al remunerar a los desarrolladores por el uso que se haga de algunos de estos servicios dentro de sus aplicaciones.

En la Figura 2-1 se presenta una imagen esquemática de la estructura de servicios ofrecidos por la plataforma.

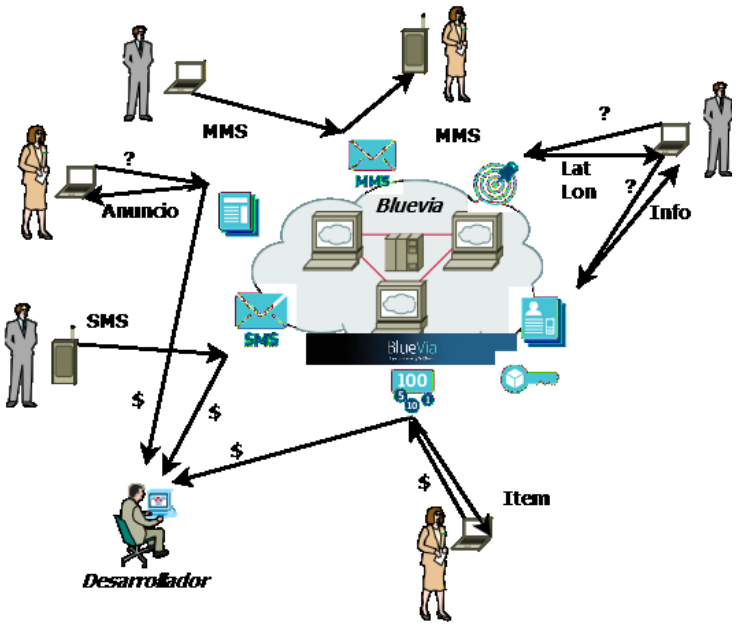


Figura 2-1 Imagen esquemática de los servicios de *Bluevia*

2.1.1 Actores en *Bluevia*

Dentro del escenario de funcionamiento de *Bluevia*, interactúan los siguientes actores:

- **Plataforma**. La plataforma es el conjunto de servidores de inteligencia de telefonía móvil, pagos y web, que permiten el funcionamiento de los servicios -tanto públicos como privados- de *Bluevia*.

  En ella, se deben dar de alta los **desarrolladores** y los **usuarios finales** para poder utilizar los API.

- **Desarrolladores**. Estos actores son quienes desarrollan las aplicaciones que invocan los servicios de los API.

  De cara a la plataforma, existen dos tipos de desarrolladores:

  - ***Generales***. Son las empresas o desarrolladores individuales que utilizan la plataforma por los cauces normales. Podrán aprovecharse de los servicios públicos -en ocasiones denominados *Non-Trusted* como contraposición a los servicios privados denominados *Trusted*-, y están obligados a que los **usuarios finales** de sus aplicaciones, las autoricen explícitamente de cara a la plataforma a usar los servicios.

  - ***Trusted-Partners***. Grandes compañías de crédito reconocido –como ***Google*** o ***Electronic Arts,*** **entre otros**- con un acuerdo particular con ***Telefónica*** que les permite acceder a los servicios privados -Trusted- de *Bluevia*, y que no necesitan la autorización explícita del **usuario final** de cara a la plataforma -aunque el acuerdo con ***Telefónica*** obliga a las compañías a informar a los usuarios de las condiciones de *Bluevia*-.

  Tanto generales como *Trusted*, todos los desarrolladores pueden incluir los servicios de *Bluevia* de forma gratuita en sus aplicaciones.

- **Usuarios finales**.

  Cuando se habla de usuario de los servicios en *Bluevia*, generalmente se habla de una aplicación que invoca uno o más servicios de la plataforma.

  Sin embargo, el término "**Usuario final**", se refiere a la persona física que autoriza a las aplicaciones a invocar los servicios en su nombre, y que llegado el caso pagará por el uso de estos.

  En la Figura 2-2 se presenta una imagen esquemática de las relaciones entre los actores de la plataforma.

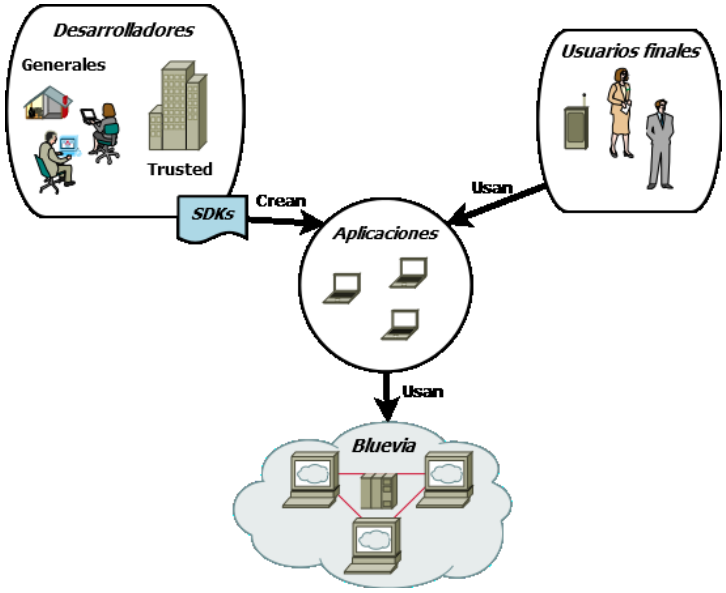


Figura 2-2 **Imagen esquemática de las relaciones entre los actores de *Bluevia***

2.1.2 Puntos de acceso de la plataforma

La plataforma visible de *Bluevia* está compuesta por varios *endpoints* con diferente cometido:

- **Web principal: *Portal*** *(bluevia.com)*

  Portal es la web principal de *Bluevia*. Puede verse un fragmento de la página principal en la Figura 2-3.

Figura 2-3 **Fragmento de la página principal del portal del *Bluevia: Portal***

Es una web informativa y de gestión, enfocada a los desarrolladores de aplicaciones, que ofrece dos funcionalidades diferenciadas:

- Información general sobre la plataforma.

  En la página se pueden encontrar entre otras cosas: documentación y recursos relacionados con los servicios ofrecidos, noticias sobre la evolución de la plataforma o eventos.

- Gestión de cuentas de desarrolladores.

  También pueden darse de alta los desarrolladores que deseen usar los servicios de la plataforma en sus aplicaciones.

  Una vez creada la cuenta de desarrollador, las acciones principales que este podrá realizar son:

  -Gestionar sus datos: personales y bancarios.

  -Dar de alta y gestionar aplicaciones: conseguir datos y créditos para pruebas, lanzar las aplicaciones en *AppStores* o seguir las estadísticas de uso y ganancias.

- **Web de autorización de aplicaciones: *Connect*** *(connect.bluevia.com)*

Este punto de acceso es el encargado de ofrecer al usuario final una interfaz web para darse de alta en los servicios de *Bluevia*, autorizar a las aplicaciones a utilizar los servicios públicos en su nombre, gestionarlas, y controlar los gastos que puedan producirse al utilizarlas. Puede verse un fragmento de la página principal en la Figura 2-4.

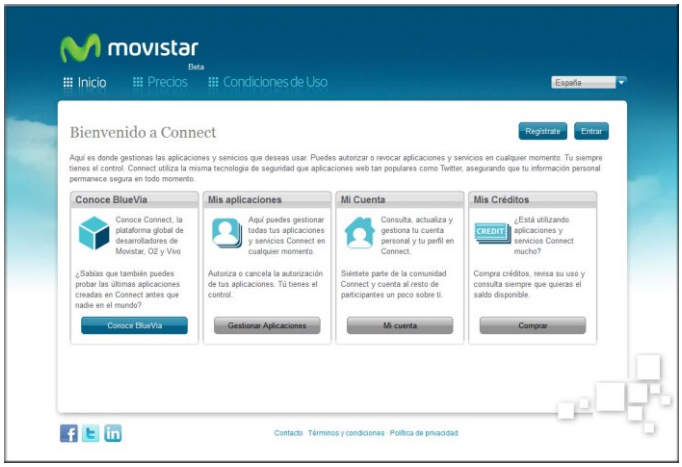


Figura 2-4 **Portada del portal de autorización de *Bluevia* para España: *Connect***

A pesar de que el punto de acceso es único, el portal web es dependiente de cada país, ya que en cada uno imperarán diferentes condiciones legales y de facturación. Por ejemplo en España este portal será de *Movistar*, con condiciones legales y tarifas españolas; mientras que en Reino Unido el portal será de O2, con sus respectivas condiciones legales y tarifarias.

- **Servicios públicos** *(api.bluevia.com/services)*

  Este es el *endpoint* que ofrece los servicios públicos de *Bluevia.* Se subdivide en dos tecnologías de invocación: REST y RPC. Cada una de ellas se subdivide progresivamente en API, servicio y opciones particulares.

- **Servicios privados**

  De la misma forma que en el caso público, el *endpoint* privado se subdivide progresivamente en: tecnología de invocación, API, servicio y opciones particulares.

2.1.3 Aplicaciones e identificadores

Cuando una aplicación hace uso de los servicios ofrecidos por Bluevia, esta ha de identificarse. Así mismo, ha de identificarse al usuario final de la aplicación en los servicios que lo requieran.

Para el caso de los servicios públicos, como se ilustra en la Figura 2-5, *Bluevia* usa dos juegos de credenciales pertenecientes al esquema de *OAuth*[11]:

- Los ***Consumer Keys*** -denominados *"APIKeys"* en Bluevia-, identifican directamente a las aplicaciones.

  Consisten en dos cadenas alfanuméricas únicas, identificador público y clave privada, que son generadas por *Bluevia* en el momento que un desarrollador da de alta una nueva aplicación en el portal.

  - El identificador público se envía en cada petición realizada por la aplicación a *Bluevia*, dentro de la cabecera HTTP de autorización OAuth -con la clave: *oauth_consumer_key*-. De modo que la plataforma pueda saber, que aplicación está invocando los servicios.

  - La clave privada se usa para verificar, que la aplicación que accede a los sistemas, no es una aplicación ilegítima haciéndose pasar por la original. Se utiliza para el cifrado de la firma de la cabecera de HTTP de autorización OAuth -oauth_signature-.

- Los ***Token Credentials*** -o simplemente ***Tokens***- identifican indirectamente a los usuarios finales, para cada aplicación.

  Son otro par de cadenas alfanuméricas codificadas en Base64, que simbolizan la autorización por parte del usuario final a que la aplicación realice peticiones en su nombre.

  Se crean típicamente durante el primer uso de la aplicación por parte del usuario final; en un proceso guiado en el que el usuario ha de aceptar las condiciones de *Bluevia* en la web de *Connect*.

  En algunas ocasiones, como en el caso del API de *Payment*, estos *Tokens*, no identifican la autorización dada por un usuario final a una aplicación en general, si no que más concretamente y para aportar más seguridad, identifican la autorización dada

por un usuario para realizar una sola transacción en particular. En este caso los *Tokens* se consideran de un solo uso efectivo –*"one shot Tokens"*-.

Estos identificadores también se componen de parte pública y privada, para evitar suplantaciones.

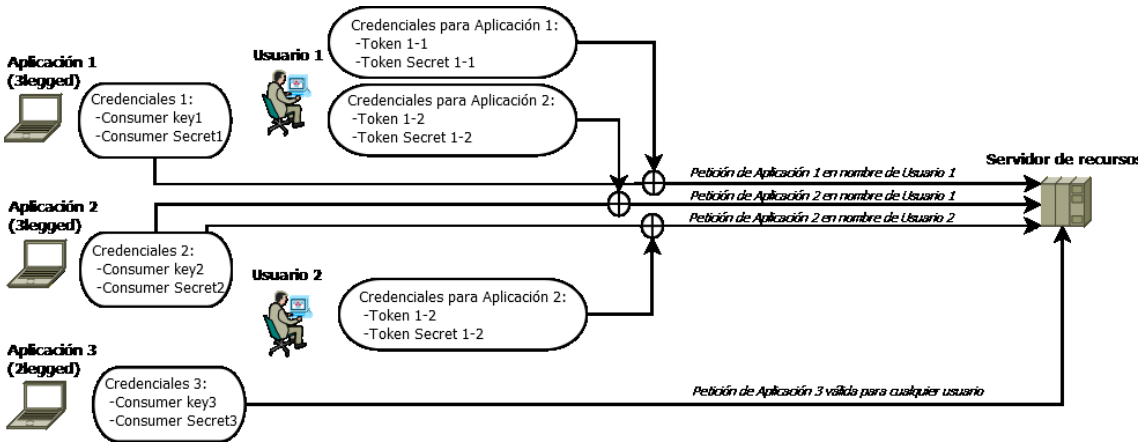


Figura 2-5 Esquema de credenciales de *Bluevia* para los servicios de acceso público

Para el caso de los servicios privados, como se ilustra en la Figura 2-6, *Bluevia* utiliza el mismo tipo de credenciales para las aplicaciones, que en el caso público; pero con un esquema diferente de identificación para los usuarios:

- Certificado SSL del *Trusted-Partner* + MSISDN. El *Trusted-Partner* garantiza que el usuario final ha aceptado las condiciones de *Bluevia*, proveyendo su certificado SSL en las conexiones con los servicios privados; e identifica al usuario a través del MSISDN asociado a su nombre.

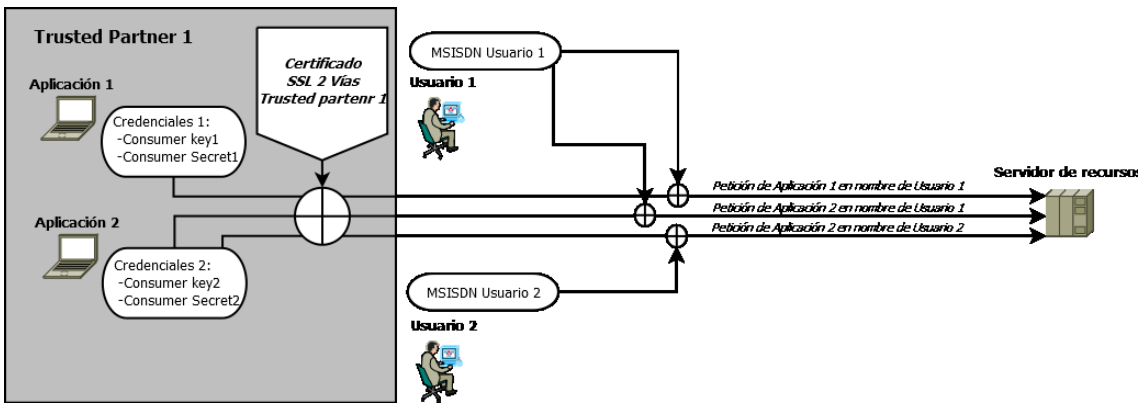


Figura 2-6 Esquema de credenciales de *Bluevia* para los servicios de acceso privado

En adelante se hablará de dos modos de autorización/autenticación distintos para el acceso a los servicios:

- ***Two-legged*** *-2legged-*. Este modo se refiere al esquema de autorización/autenticación en el que solo se identifica, usando *OAuth*, a la aplicación que hace uso del servicio; y

por lo tanto, como puede apreciarse en la cabecera de la Figura 2-7, prescinde de *Tokens*.

Esto es debido a que el servicio en uso no necesita el consentimiento del usuario final para funcionar -ya que el servicio es gratuito y no se vulnera su privacidad, o el consentimiento ya ha sido aportado por otros medios fuera del esquema *OAuth*-.

Algunos servicios públicos, y todos los privados usarán este esquema.

```
Authorization: OAuth oauth_consumer_key=" Credencial_de_aplicación
",oauth_nonce="Nonce",oauth_signature_method="HMAC-
SHA1",oauth_version="1.0",oauth_timestamp="1359911804",oauth_signature="Firma"
```

Figura 2-7 Imagen de cabecera HTTP *Authorization:OAuth 2legged*, sin credenciales de usuario, opcionalmente puede aparecer el campo vacío

- ***Three-legged*** *-3legged-*. Este modo se refiere al esquema donde tanto la aplicación como el usuario final que hacen uso del servicio, tienen que ser identificados usando *OAuth*; y por lo tanto los Tokens son obligatorios. Puede apreciarse en la cabecera de la Figura 2-8.

  La mayoría de los servicios de acceso público usarán este esquema.

En el punto del Estado del Arte: 2.3, se ampliará la información sobre *OAuth*.

```
Authorization: OAuth
oauth_token="Credencial_de_usuario",oauth_consumer_key="Credencial_de_aplicación",o
auth_nonce="Nonce",oauth_signature_method="HMAC-
SHA1",oauth_version="1.0",oauth_timestamp="1359911829",oauth_signature="Firma"
```

Figura 2-8 Imagen de cabecera HTTP *Authorization:OAuth 3legged*

2.1.4 Modos de uso operativo

Dado que *Bluevia* es una plataforma de servicios dirigidos al desarrollo de aplicaciones, ofrece tres modos de uso operativo:

- ***Sandbox***. Este modo de uso de los servicios es el apropiado para los primeros estadios de desarrollo de una aplicación. Es una "caja de arena" que controla que los datos de las peticiones sean correctos, y devuelve respuestas genéricas en consecuencia. La invocación de los servicios será sobre una URL modificada -APIName_Sandbox, salvo en el caso de OAuth- y siempre gratuita, pero nunca ejecutarán acciones reales.

  Para poder autorizar a una aplicación para que use los servicios de este modo, el usuario tiene que tener cuenta de desarrollador en el Portal, y completar el proceso de *OAuth* en la sección de "*test-apps*" de este.

  Resumiendo en palabras de *Bluevia*: *"…ofrece exactamente la misma experiencia que el entorno Live, excepto que no se genera tráfico en la red real…"*.

- ***Testing***. Este modo se ofrece para realizar demostraciones, o probar una aplicación cuando está prácticamente terminada. Ejecuta todas las acciones reales al invocar un servicio, pero solo sobre los datos del propio desarrollador de la aplicación. En vez de

dinero real, consume créditos que son ofrecidos todos los meses de forma gratuita por la plataforma, o pueden comprarse en packs especiales.

Como en el caso de *Sandbox*, este modo no está enfocado al usuario final, sino a desarrolladores, por lo que la autorización debe de completarse también en *"test-apps"* de Portal.

- ***Live***. Este es el modo enfocado al uso comercial de una aplicación. Tras invocar los servicios, ejecuta acciones reales sobre la información del usuario final que lo haya autorizado; y tarifica con moneda real.

  A diferencia de los otros dos modos, este no está enfocado a pruebas de desarrollador, por lo que el usuario deberá autorizar a la aplicación en "*Connect*" de la web de su operador.

En la Figura 2-9 se muestra una tabla con las diferencias entre los modos de uso, más importantes de cara al desarrollador:

Modo	***¿Ejecución real del servicio?***	***El usuario final autoriza en:***	***Costo:***
Sandbox	No, aunque se comporta coherentemente. URL Servicio: /APIName_Sandbox/.	"Test-apps" en Portal.	Gratuito.
Test	Si, totalmente. URL Servicio: /APIName/.	"Test-apps" en Portal.	Créditos (gratuitos y comprables).
Live	Si, totalmente. URL Servicio: /APIName/.	"Connect" en web del operador.	Moneda de curso legal.

Figura 2-9 Tabla de diferencias entre los modos de uso de *Bluevia*

2.1.5 APIs de *Bluevia*

Se define como API de *Bluevia* a un conjunto de servicios con una temática común, "invocables" mediante peticiones HTTP, y a los objetos y reglas que los definen.

Bluevia ofrece varios API[2] que contienen uno o más servicios que pueden ser invocados de forma tanto pública, como privada -con varias ventajas sobre el acceso público-:

- Selección y recuperación de anuncios en formato imagen o texto: ***Advertising API***.
- Selección y recuperación de información asociada a un usuario de la red móvil -personal, contrato, acceso y contrato-: ***User Context API***.
- Recuperación de la localización por redes de telefonía, de un dispositivo móvil: ***Location API***.
- Envío y recepción de mensajería SMS y MMS: ***SMS y MMS API***.
- Pagos de bienes -materiales o virtuales- a través de la factura telefónica: ***Payment API***.

Además, ofrece un API público para autorizar el uso de los anteriores: ***OAuth API***; y varios servicios no enfocados directamente a mejorar la experiencia del usuario final, sino a mejorar la del desarrollador.

En la Figura 2-10 se presenta una tabla informativa de la disponibilidad de estos servicios por país:

Figura 2-10 Tabla de disponibilidad de servicios de *Bluevia* por país. Fuente: *Bluevia*

Nota: Puede encontrarse más información sobre estos servicios ofrecidos por cada API en el anexo: 7.3 DESCRIPCIÓN DE LOS SERVICIOS OFRECIDOS POR *BLUEVIA*.

Los API están disponibles en varios *endpoints*, compuestos de la siguiente forma:

https://bluevia.com/services/*tipo de acceso HTTP***/***Nombre del API***/**

Y a partir de ahí se montan los servicios, por ejemplo:

https://api.bluevia.com/services/REST/Advertising/simple/requests...

Cada uno de los servicios requiere la información de autorización para funcionar, además de varios datos particulares relacionados con lo que ofrezca. Estos datos están descritos en las especificaciones de cada servicio, y en general, también vienen descritos en archivos XSD que pueden facilitar la labor de invocación.

Por ejemplo para el caso de la petición para descargar un anuncio, es necesaria información sobre el país en el que se está demandando dicho anuncio; y además existe la posibilidad de afinar la búsqueda con varios parámetros extra que deberán viajar en formato *FormUrlEncoded* dentro del cuerpo de la petición, como se muestra en la captura de tráfico de la Figura 2-11.

```
POST
https://api.bluevia.com/services/REST/Advertising/simple/requests?version=v1
Authorization: OAuth
oauth_token="token",oauth_consumer_key="key",oauth_nonce="*",oauth_signature_method="HMA
C-SHA1",oauth_version="1.0",oauth_timestamp="*",oauth_signature="*"
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: api.bluevia.com
Content-Length: 158

ad_presentation=0101
&ad_request_id=ad3f0f598ffbc660fbad9035122eae7404%2F10%2F2012%2017%3A20%3A02
&ad_space=BV15125
&country=ESP
&keywords=Bluevia
&user_agent=none
```

Figura 2-11 **Captura de una petición para la descarga de un anuncio, en la que pueden apreciarse los parámetros de búsqueda**

Con esta información *Bluevia* ejecutará la búsqueda del anunció deseado, y devolverá una respuesta acorde con la información demandada. Esto puede observarse en la Figura 2-12.

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=UTF-8
Content-Length: 630
Date: Thu, 04 Oct 2012 17:20:01 GMT

<?xml version="1.0" encoding="UTF-8"?>
<NS1:adResponse xmlns:NS1="http://www.telefonica.com/schemas/UNICA/REST/sgap/v1/"
id="111122010-10-18T10:00:0030c8db243ad" version="2">
<NS1:ad id="52891020" ad_placement="1" campaign="52891004" flight="52891019">
<NS1:resource ad_presentation="0101">
   <NS1:creative_element type="image">
          <NS1:attribute type="locator">
              https://api.bluevia.com/services/Ad/BlueViaBanner.png
           </NS1:attribute>
           <NS1:interaction type="click2wap">
               <NS1:attribute type="URL">
                   http://www.bluevia.com
               </NS1:attribute>
           </NS1:interaction>
    </NS1:creative_element>
</NS1:resource>
</NS1:ad>
</NS1:adResponse>
```

Figura 2-12 **Captura de una respuesta, con datos de un anuncio acorde a los requisitos demandados previamente.**

Nota: Para una descripción más extensa de los servicios existentes, y sus requisitos, véase el anexo: 7.2 RESUMEN DE LAS ESPECIFICACIONES PARA EL ACCESO A LOS SERVICIOS DE *BLUEVIA*.

2.1.6 Los SDK de acceso a los servicios de *Bluevia*

Aunque la documentación de acceso a los servicios de *Bluevia* fue publicada desde el primer día; dada la cantidad de requisitos existentes, resultaba relativamente compleja de implementar para alguien ajeno a la plataforma.

Para facilitar ese proceso, nacieron a la vez que la plataforma los SDK de acceso a los servicios de *Bluevia*, en código abierto y licencia LGPL, como una ayuda extra -y reclamo- para que los desarrolladores pudiesen comenzar a usar rápidamente los servicios.

Los SDK, ofrecían inicialmente el código de acceso a los servicios y las librerías generadas -cuando correspondiese-, en ***Java***[3], ***PHP***[4], ***Ruby***[5], y ***C#*** para ***.NET***[6] -que es el caso que nos ocupa-.

A partir de la versión 1.2 de la plataforma, aparecen también y un *framework* completo de desarrollo para el ***Visual Studio 2010*** -fruto del *parternship* con ***Microsoft***- y el SDK para ***Android***[7].

Estos SDK han ido evolucionando con los cambios que han surgido en la plataforma durante el tiempo: se ha añadido nuevos API, nuevos servicios, y nuevos esquemas de acceso. Y en el ámbito de la actual versión 1.6 se han ampliado con el SDK de ***Arduino***[8], y varias versiones no oficiales -***ObjetiveC, Python, entre otras***-, todas disponibles en el repositorio ***GitHub***[24] de *Bluevia*[9].

2.1.7 Limitaciones de las versiones anteriores del SDK para *.NET*

Por problemas de plazos de entrega, hasta la versión 1.5 -inclusive- de *Bluevia*, el SDK de ***.NET*** tuvo que irse ampliando sobre una estructura ideada para la versión 1.0 de la plataforma. Dicha estructura inicial, no contemplaba más que un par de API, y solo acceso a servicios públicos *3-leeged*; por lo que la evolución en el tiempo del código fue encontrando varias dificultades en sus cimientos:

- En las versiones anteriores del SDK, los diseños y estructuras obligaban al desarrollador a cargar un solo cliente monolítico de *Bluevia* -con toda la funcionalidad y objetos disponibles-, aunque solo necesitase la funcionalidad de un API en su aplicación.

- Esas versiones dependían también de la Librería externa *DotNetOpenAuth*, para gestionar la autenticación de usuarios y aplicaciones; elemento que aunque facilitaba la labor en los casos más generales, en otros casos afectaba negativamente:

    - Complicando la depuración del desarrollo, al gestionar internamente la librería los sockets de comunicación.

    - Sobrecargando el SDK con opciones de funcionalidad innecesarias.

    - Obligando a realizar soluciones poco ortodoxas para poder cumplir las especificaciones particulares que *Bluevia* añadía a los requisitos de autenticación.

    - Y haciendo incompatible gran parte del código del núcleo de las versiones del SDK de acceso público y privado.

- El código interno de inicialización de esas librerías, era muy complejo; y su configuración operativa, difería por completo tanto en forma como en funcionalidad, de la de los SDK del resto de lenguajes.

  En los diferentes lenguajes, los clientes se instanciaban de forma similar y mantenían una filosofía de uno por API, pero como se muestra en la Figura 2-13, diferían tanto las estructuras de los SDKs, como las invocaciones a los métodos.

  Mientras que en el caso de ***C#***, como se muestra en la Figura 2-14, ni siquiera se mantenía la filosofía de cliente.

Instanciación de cliente en los antiguos SDK de *Ruby*:

```
@bc = Bluevia::BlueviaClient.new(
  { :consumer_key   => ”The Key”,
    :consumer_secret=> ”The Secret”,
    :token         => @token,
    :token_secret   => @token_secret,
    :uri          => “https://api.bluevia.com”,
    :mode          => “LIVE”
  })
```

Preparación e invocación de un método del cliente:(Se invoca directamente con parámetros)

```
@advertising = @bc.get_service(:Advertising)
ad = @advertising.request({
  :user_agent => ”Mozilla 5.0“
  :ad_presentation => ”0104“
  :ad_request_id => “Ad Request Id“
  :ad_space => “1200”
  :keywords => “sports“
  :protection_policy => “1“
})
```

Instanciación de un cliente en los antiguos SDK de *Java*:

```
OAuthToken consumer = new OAuthToken("The Key ", " The Secret ");
OAuthToken accesstoken = new OAuthToken("token "," token_secret ");

Advertisement adclient = new Advertisement(
  consumer,
  accesstoken,
  Mode.LIVE
);
```

Preparación e invocación de un método del cliente:(Se crea previamente un objeto autogenerado para pasárlo como parámetro)

```
SimpleAd sa = new SimpleAd();
  sa.setAdreqId("Ad Request Id ");
  sa.setUserAgent("Mozilla 5.0");
  sa.setAdSpace("1200");
  sa.setProtection_policy("1");

Adresponse response = adclient.send(sa);
```

Figura 2-13 Código comparativo de la creación de un cliente y la instanciación de uno de sus métodos, entre los SDKs de *Ruby* y *Java,* para su versión 1.5

Instanciación de un cliente en los antiguos SDK de *.NET*:(Se configuran en la librería las credenciales de la aplicación, el modo de operación, y tras ello se instancia un cliente monolítico)

```
Bluevia.Core.Configuration.Client.setConsumer("The Key ");
Bluevia.Core.Configuration.Client.setConsumerSecret("The Secret ");
Bluevia.Core.Configuration.Client.setEnvironment(Bluevia.Core.Configuration.ENVIRONMENT.LIVE)
BlueviaClient client = new BlueviaClient();
```

Preparación e invocación de un método del cliente:(Se crea una petición genérica sobre el cliente, tras ello, se configuran en la librería las credenciales de usuario, y se autentica la petición)

```
IBlueviaClient request = client.CreateRequest();
Bluevia.Core.Configuration.Client.setTokenPair("AccessToken", "SecretToken");
IBlueviaClient authenticatedRequest = request.AuthenticateClient();
```

(Se crea un objeto autogenerado para pasárlo como parámetro, y se envía, en una petición de servicio, sobre la petición genérica autenticada)

```
Bluevia.SGAP.SimpleAdRequest adRequest = new SimpleAdRequest(
  adRequestId, adSpace, userAgent, protectionPolicy,
  adRequestId,targetUserId,"country"*optional, adPresentationSize);
var response = authenticatedRequest.SGAP.AdRequest.Send(adRequest,false);
```

Figura 2-14 Código de la creación de un cliente y la instanciación de uno de sus métodos, para la versión 1.5 del SDK de *.NET*

- Además, la lógica del sistema acababa centralizada en un método extremadamente grande y complejo, perteneciente a una clase también difícil de manejar.

Todo esto, hacía que desarrollar nuevas funcionalidades no resultase fácilmente abordable, y que la ejecución del código comenzase a ser ligeramente pesada

En la Figura 2-15 se muestra una imagen de la estructura general de los SDK previos a la versión 1.6.

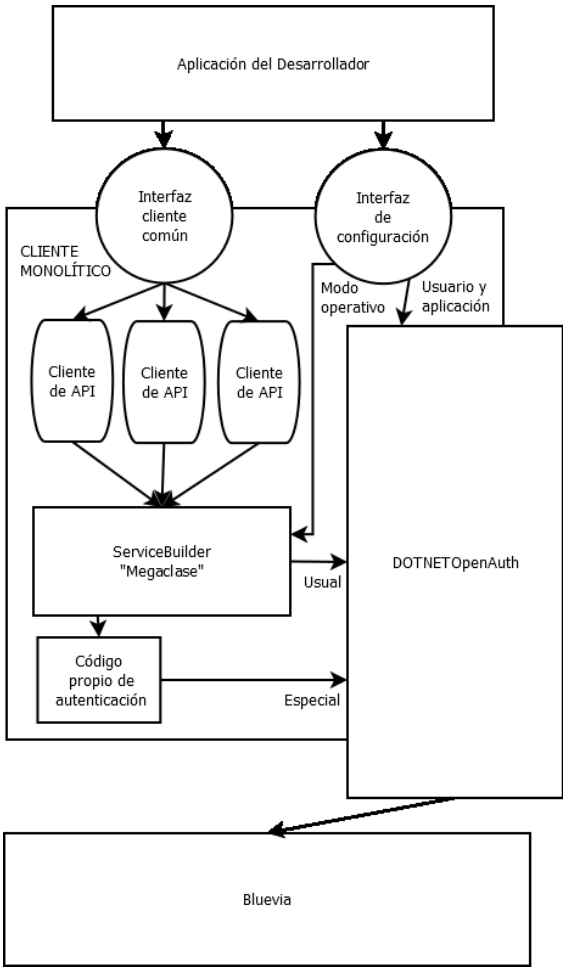


Figura 2-15 Estructura general del SDK previo a la versión 1.6

Nota: Con el visto bueno, y los plazos necesarios para "refactorizar" en profundizad el SDK, la versión 1.6 procura escapar de estas limitaciones.

2.2 APIs WEB

Si una interfaz de programación de aplicaciones -API por su siglas en Inglés- “, es un conjunto de rutinas que provee acceso a las funciones de un determinado *software*“[37], una API Web, es un conjunto de rutinas que provee acceso a funciones de servicios generalmente remotos, a través de un interfaz de acceso a la web.

Estas API son de gran utilidad para los desarrolladores de aplicaciones, ya que pueden ser invocadas sin la necesidad de cargarlas en las propias aplicaciones, o crear protocolos de acceso específicos; tan solo es necesario poseer un sistema de acceso HTTP.

Además, aunque no necesariamente, el acceso a estas API suele ser especificado de forma pública y generalmente gratuita, por lo que de esta forma los desarrolladores pueden ofrecer múltiples funcionalidades de diverso carácter -que por lo general conllevarían una infraestructura importante-, que son actualizadas y mantenidas de forma transparente, con un coste en desarrollo muy bajo.

Como dato de interés para ilustrar este potencial, a finales de 2012 el número de estas API contabilizado por el portal ***ProgrammableWeb*** es de alrededor de 8400; muchas de ellas, proporcionando una funcionalidad tan útil como las de ***Google***[42] o ***Amazon***[43].

Nota: Puede existir cierta tendencia a usar los términos API Web y *Web Service* de forma indistinta, pero el *W3C*[39] define que los *Web Services* emplean SOAP[40] como protocolo de acceso -SOAP emplea un subconjunto de funcionalidades de HTTP-, por lo que los *Web Services* estarían embebidos dentro de la generalidad de los APIs Web.

Muchas de estas API, se ofrecen mediante un simple interfaz HTTP, aunque otras se exponen a través de sistemas más complejos -SOAP, ***JavaScript***…- que finalmente terminan recayendo en HTTP.

En *Bluevia* puede accederse a las funcionalidades ofrecidas a través de las API Web expuestas. Según el servicio que vaya a ser accedido, la comunicación usará una de las siguientes tecnologías:

- **Transacciones HTTP simples**, en los casos en los que en la comunicación existan cuerpos *Multipart, o* no se necesiten enviar objetos.
- **SOAP**, en los casos en los que las respuestas en la comunicación, deban contener objetos definidos en un sobre SOAP.
- **XML-RPC**, en los casos en los que las transacciones sean realizadas para ejecutar algún proceso remoto.

2.2.1 HTTP

El protocolo de transferencia de "hipertexto" -HTTP por sus siglas en inglés [38]-, fue diseñado para la transferencia de archivos HTML; iniciando lo que hoy en día es la *World Wide Web.*

En resumen, es un protocolo sin estado que funciona sobre el protocolo de transporte TCP, y consiste en transacciones petición-respuesta en NVT ASCII. Estas están formadas por un encabezado, un salto de línea, y generalmente un cuerpo.

Las peticiones pueden ser de 8 tipos con distintos cometidos. Por ejemplo GET, pretende simplemente recuperar un recurso completo; HEAD, pretende recuperar la cabecera del recurso con la intención de conocer si este ha sido modificado y DELETE, pretende eliminar el recurso.

Los tipos de respuestas vienen definidos en 5 bloques de códigos numéricos de tres cifras: 1XX, para respuestas provisionales; 2XX, para respuestas exitosas; 3XX, para respuestas de redirección; 4XX, para errores cometidos en la petición; 5XX, para errores en el servidor.

Los encabezados, contienen cabeceras con información relativa al mensaje que transmiten, o relativa a la información que se espera recibir. Por ejemplo una cabecera *Authorization* puede proporcionar credenciales para acceder a un recurso, mientras que una cabecera *Content-Type* especifica el tipo de contenido que se transmite en el cuerpo.

Los cuerpos en las peticiones contienen información de entrada para algún proceso del servidor; mientras que en las respuestas contienen la información solicitada.

Esta información puede ir codificada dentro de los cuerpos en varios formatos, de acuerdo a los tipos de medios de Internet[44]. Por ejemplo con un tipo *application/xml*, el cuerpo será un documento XML; con un tipo *multipart/form-data*, el cuerpo contendrá varios bloques de datos de distintos tipos de medio, etc.

Puede verse como ejemplo de las partes que componen una transacción HTTP la imagen de la Figura 2-16.

Petición: Compuesta solo de encabezado.
1: Tipo de petición y recurso URL solicitado.
2: Cabeceras de la petición.

Respuesta: Respuesta exitosa, compuesta por encabezado y cuerpo.
3: Cabeceras de la respuesta.
4: Recurso solicitado durante la petición, en el cuerpo de la respuesta en formato XML.

Figura 2-16 Imagen de las partes de una transacción HTTP, y leyenda

Existe una versión segura de HTTP denominada HTTPS, que funciona sobre la capa de adaptación criptográfica SSL/TLS, y cambia de número de puerto, pero que por lo demás, funciona igual que la original.

2.2.2 HHTP para *Bluevia*

Centrándonos en el caso de *Bluevia*, el acceso a las API ofrecidos se realiza a través de HTTPS, y se hace uso de tres de los tipos de peticiones disponibles en HTTP.

- Peticiones ***Get***: Se utilizan para acceder a los servicios que se basan en una consulta de información. Por ejemplo, para la recuperación de la localización de un terminal móvil, o la recuperación del estado de envío de un SMS. En ellas puede enviarse información acerca del recurso solicitado, en forma de parámetros en la URL.

- Peticiones ***Post***: Se utilizan para los casos en que los servicios requieran que se les envíe información de entrada para completar su función -incluyendo las operaciones SOAP y XML-RPC-. Por ejemplo, para el envío de un mensaje SMS, o el pago de una transacción. La información de los cuerpos puede estar en formato *Multipart, FormUrlEncoded* o XML.

- Peticiones ***Delete***: Se utilizan para cancelar subscripciones activadas previamente a través de un servicio. Para los casos, de la baja en los servicios de notificaciones de mensajería.

2.2.2.1 Cabeceras HTTP en *Bluevia*

Además de las obligatorias como *Host* o *Date,* las cabeceras HTTP más importantes para las transacciones con *Bluevia* son:

- Cabecera de autorización/acceso: ***Authorization***. Esta cabecera se completa utilizando *OAuth*. La llevarán todas las peticiones para garantizar su legitimidad de cara a la plataforma.

- Cabeceras de contenido: ***Content-Type*** y ***Content-Lenght***. Son cabeceras de "metainformación", que informan sobre como procesar el cuerpo del mensaje. Definen respectivamente el tipo y el tamaño del medio insertado en el cuerpo.

  La primera de las cabeceras se complementa con el parámetro *charset*, que define el juego de caracteres en el que se encuentra codificado el medio contenido; y en el caso de que el tipo de medio sea un *Multipart*, la cabecera se complementará además con el parámetro *boundary*, que define el patrón de caracteres que delimita los diferentes bloques de datos contenidos.

- Cabecera de redirección: ***Location***. Esta cabecera de respuestas indica la URL donde se ubica un recurso relacionado con petición enviada; por ejemplo, al hacer una petición correcta para enviar un SMS, la respuesta del servidor devuelve una cabecera *Location* con la URL a la que se podrá preguntar en adelante por el estado de envío de ese mismo mensaje.

2.2.2.2 Cuerpos HTTP en *Bluevia*

Los cuerpos en las transacciones de *Bluevia* pueden ser de tres tipos principales:

- Tipo de medio ***application/x-www-form-urlencoded***. Este tipo es usado usualmente en la *Web* para enviar los datos que un usuario ha rellenado en un formulario. Como puede verse en la imagen de la Figura 2-17, consiste en una serie de pares "campo-valor" unidos en una lista, de la misma forma que los parámetros de una URL. Se utiliza en *Bluevia*, como entrada o salida de datos para un servicio, cuando no es necesaria un objeto estructurado.

```
POST https://api.bluevia.com/services/REST/Advertising_Sandbox/simple/requests?version=v1 HTTP/1.1
Authorization: OAuth
oauth_token="ad3f0f598ffbc660fbad9035122eae74",oauth_consumer_key="vw1201265450598 6",oauth_nonce="bccfbbea804c4f609d7b6
d752d5189e0",oauth_signature_method="HMAC-
SHA1",oauth_version="1.0",oauth_timestamp="1356114418",oauth_signature="rmWvfEyERm%2FFzlSDQgEulnbIB8k%3D"
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: api.bluevia.com
Content-Length: 166
Expect: 100-continue
Connection: Keep-Alive

ad_presentation=0101&ad_request_id=ad3f0f598ffbc660fbad9035122eae7421%2F12%2F2012%2018%3A26%3A57&ad_space=BV15125
&keywords=Bluevia&protection_policy=1&user_agent=none
```

Figura 2-17 Imagen de petición HTTP con cuerpo en formato x-www-form-urlencoded

- Tipo de medio ***application/xml***. Este tipo consiste en un texto formateado en XML, que simboliza un objeto estructurado de entrada o salida para un servicio; o la invocación de un método remoto.

    - ***SOAP***: En el primero de los casos, cuando el cuerpo XML simboliza un objeto, los datos que contiene van envueltos en un sobre -también XML-, y para *Bluevia* la transacción es considerada una comunicación SOAP. Se muestra un ejemplo en la imagen de la Figura 2-18.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=UTF-8
Content-Length: 754
Date: Tue, 08 Jan 2013 15:54:52 GMT

<?xml version="1.0" encoding="UTF-8"?>
<NS1:receivedMessages xmlns:NS1="http://www.telefonica.com/schemas/UNICA/REST/mms/v1/">
  <NS1:receivedMessages>
    <NS1:messageIdentifier>06603100100349234400</NS1:messageIdentifier>
    <NS1:destinationAddress>
      <NS2:phoneNumber xmlns:NS2="http://www.telefonica.com/schemas/UNICA/REST/common/v1">546780</NS2:phoneNumber>
    </NS1:destinationAddress>
    <NS1:originAddress>
      <NS3:phoneNumber xmlns:NS3="http://www.telefonica.com/schemas/UNICA/REST/common/v1">546780</NS3:phoneNumber>
    </NS1:originAddress>
    <NS1:subject>SANDBLUEDEMOS This is a Dummie MMS Subject for MMS_MO</NS1:subject>
    <NS1:dateTime>2013-01-08T15:54:52.344Z</NS1:dateTime>
  </NS1:receivedMessages>
</NS1:receivedMessages>
```

Figura 2-18 Imagen de respuesta HTTP con cuerpo en formato XML, para comunicación SOAP

    - ***XML-RPC***: En el segundo caso, el texto define un método, y los parámetros con los que se le invoca. Se está usando una llamada a procedimiento remoto -RPC en sus siglas en Inglés[45]-, usando el texto XML del cuerpo HTTP como transporte. Se muestra un ejemplo en la imagen de la Figura 2-19.

```
POST https://api.bluevia.com/services/RPC/Payment_Sandbox/payment?version=v1 HTTP/1.1
Authorization: OAuth
oauth_timestamp="1359113544",oauth_token="e788cd536f08455f3f8c54018a0fbaee",oauth_consumer_key="Gy12012656370368",oauth_
nonce="ee8a76237b294f4cbdb233749cb48121",oauth_signature_method="HMAC-
SHA1",oauth_version="1.0",oauth_signature="8xnUXCmRsdFjXOWIAJbEcnTtDgI%3D"
Content-Type: application/xml; charset=utf-8
Host: api.bluevia.com
Content-Length: 609
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0"?>
<methodCall xmlns:com="http://www.telefonica.com/schemas/UNICA/REST/common/v1"
xmlns="http://www.telefonica.com/schemas/UNICA/RPC/payment/v1">
  <id xmlns="http://www.telefonica.com/schemas/UNICA/RPC/definition/v1">d76226f</id>
  <version xmlns="http://www.telefonica.com/schemas/UNICA/RPC/definition/v1">v1</version>
  <method>PAYMENT</method>
  <params>
    <paymentParams>
      <timestamp>2013-01-25T11:32:24</timestamp>
      <paymentInfo>
        <amount>15</amount>
        <currency>EUR</currency>
      </paymentInfo>
    </paymentParams>
  </params>
</methodCall>
```


Figura 2-19 Imagen de petición HTTP con cuerpo en formato XML, para comunicación XML-RPC

- Tipo de medio ***multipart/form-data***: Este caso se utiliza para enviar directamente archivos, que pueden ser simples textos, pero que en su mayoría serán los bytes de archivos multimedia. Se muestra un ejemplo en la imagen de la Figura 2-20.


```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: multipart/form-data; boundary=MIMEBoundaryurn_uuid_636C752383BEC3AFDD1222262665928;
start="<0.urn:uuid:636C752383BEC3AFDD1222262665929@apache.org>"
Content-Length: 2195
Date: Tue, 08 Jan 2013 15:54:52 GMT


--MIMEBoundaryurn_uuid_636C752383BEC3AFDD1222262665928
Content-Disposition: form-data; name="root-fields"
Content-Type: application/xml;charset=UTF-8
Content-ID: <0.urn:uuid:636C752383BEC3AFDD1222262665929@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
<NS1:message xmlns:NS1="http://www.telefonica.com/schemas/UNICA/REST/mms/v1/"><NS1:address><NS2:phoneNumber xmlns:NS2
="http://www.telefonica.com/schemas/UNICA/REST/common/v1">546780</NS2:phoneNumber></NS1:address><NS1:originAddress>
<NS3:phoneNumber xmlns:NS3="http://www.telefonica.com/schemas/UNICA/REST/common/v1">546780</NS3:phoneNumber>
</NS1:originAddress><NS1:subject>SANDBLUEDEMOS This is a Dummie MMS Subject for MMS_MO</NS1:subject></NS1:message>
--MIMEBoundaryurn_uuid_636C752383BEC3AFDD1222262665928
Content-disposition: form-data; name="attachments"
Content-Type: multipart/related; boundary=BbC04y

--BbC04y
Content-Type: text/plain;charset=UTF-8
Content-Transfer-Encoding: binary
Content-ID: <2>

Optional text attachment
--BbC04y
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <3>

□□□□□□JFIF□□□□□d□d□□□□□□Ducky□□□□□□□$□□□□□□Adobe□d□□□□□□□□□□□
      □

□□


□□□□□□□□□□□□□□□□□□□□□□□□ " □)),,));::::;@@@@@@@@@@□□

□□□□□□□□□□□□□□□□□□ □□□□□ )□□□□)%'""""'%--))--99699@@@@@@@@@@@□□□□□□x□x□□"□□□□□□□□□□}
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□!1□□AQaq□"2□RS□□□□□□□□□□□□□□□□□□□□□□□□□□□!
□1AQa"q□23□□□□
□□□□□□□□?□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□v□□□□□□Z□♦WeT♦♦K♦KO♦♦r♦♦wo♦♦B♦ ♦M♦♦□]□□□□q□Ż+3k□□□□:
|□□□□□.□□□9Nv□□'♦♦m♦eb♦KO♦*n♦1♦♦5I♦D♦>S♦♦♦A♦x♦♦♦♦♦q♦;♦♦♦E♦♦
t♦>g♦♦♦□Q♦□♦♦♦♦A+♦♦z/♦♦♦♦□Y♦♦♦♦Ur♦f□♦Uq♦□♦♦%♦♦♦♦♦♦♦♦a'w□+♦♦|♦L♦}♦♦I□♦♦♦fqou>m5♦♦-♦♦?I♦□□♦
(♦w♦♦♦\L{□♦s♦g□q♦j♦D♦d♦♦♦C[♦g7□!d♦♦|c*Q♦♦(♦M♦T♦♦♦
□v.□^□□?□□□□sp□□c^□□□+y□;□□□□c□□Q□+I$□□U□□O□□□\Z"#5□□□□=□s□M□Z□□
□-□□h□ □□□□□?VD□Wv@□□□□□□□□v~□+?□□□□a□xqP□□□J1□□Y?□A□s1□□F□□□□□□sY□□E□v□+;
□
□m□□$Tm□&□□b□□VMvGX□r□□□vj□□x□□Rs□□□5%□□:□@□+x□□□□□□x□xF+nu□□□4□□m□mB♦(♦J1♦      ♦D♦♦♦-
♦♦lv□♦♦□Y♦♦♦m5□♦♦♦o♦V□7♦♦&e□v♦♦-
□♦♦t♦H♦"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
--BbC04y--

--MIMEBoundaryurn_uuid_636C752383BEC3AFDD1222262665928--
```


Figura 2-20 Imagen de respuesta HTTP con cuerpo en formato multipart, con 3 archivos contenidos: un XML, un texto, y una imagen

Los tipos de archivos soportados son: *text/plain*, *image/jpeg*, *image/bmp*, *image/gif*, *image/png*, *audio/amr*, *audio/midi*, *audio/mp3*, *audio/mpeg*, *audio/wav*, *video/mp4*, *video/avi*, *video/3gpp*.

2.3 *OAUTH*

OAuth[11] es un protocolo abierto, de autorización y autenticación segura, para servicios que usen HTTP como tecnología de comunicación. Parafraseando el sumario de la RFC:

1.- **Autenticación**. *"OAuth proporciona a clientes -clientes HTTP con una funcionalidad no exclusivamente de navegación web-, un método para acceder a recursos de servidor, en nombre del propietario del recurso -típicamente un usuario final-".*

2.- **Autorización**. *"También proporciona a los usuarios finales, un proceso para autorizar a terceras partes -en general los clientes HTTP- a acceder a sus recursos de servidor, sin compartir sus credenciales -típicamente, usuario y contraseña-".*

En la Figura 2-21 se presentan dos esquemas que ilustran las funciones descritas por el RFC de *OAuth*.

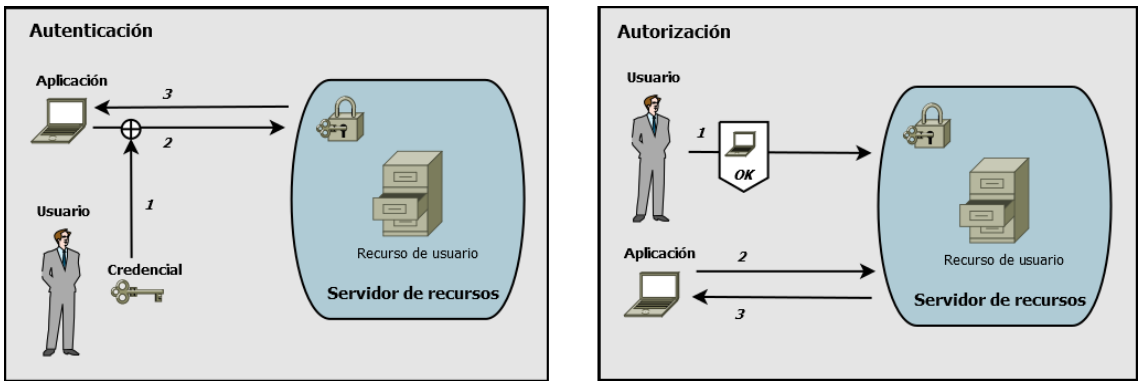


Figura 2-21 Imagen esquemática de las funciones ofrecidas por OAuth

En *OAuth* las credenciales son parejas compuesta por un identificador único, y una clave compartida relativa al primero. Se definen tres tipos:

- **Credenciales de cliente**. "*Consumer Keys",* que identifican y autentican al cliente que realiza las peticiones.
- **Credenciales temporales**. Que identifican y autentican la petición de autorización.
- **Tokens**. "*Token Keys"*, que identifican y autentican el acceso autorizado por parte del propietario del recurso.

De las dos funcionalidades que ofrece la tecnología, se describirá en primer lugar la relativa a la autenticación; ya que es la de concepto típicamente más asequible por consistir simplemente en calcular los valores de un grupo de parámetros de *OAuth*, y rellenar con ellos la cabecera HTTP *"Authorization"*, o en su defecto, el cuerpo de la petición o parámetros de URI.

Los nombres de los parámetros comienzan por el prefijo "oauth_", y junto con sus valores, son sensibles a las mayúsculas.

Se definen parámetros para contener las credenciales de los demandantes de la petición, una marca temporal y la firma codificada de la petición entre otros.

Según las credenciales existentes en el grupo de parámetros, se exponen dos tipos de autenticación:

- Una, que permite al servidor que guarda los recursos, comprobar que la petición recibida ha sido efectuada por un cliente explícitamente autorizado por el dueño del recurso.

  Identificando a ambos actores -cliente HTTP y usuario final dueño del recurso- con sus identificadores únicos.

- Y otra, para el caso en el que el recurso, no sea de la propiedad del usuario final; y en la que el servidor solo tenga que comprobar que el cliente esté autorizado a acceder a dicho recurso.

  Donde solo se identificará al cliente en la cabecera.

Nota: A pesar de que no aparecen los siguientes términos en el RFC, la primera forma de autenticación es conocida comúnmente como 3-legged, mientras que la segunda se conoce como 2-legged.

La firma es el parámetro que proporciona la seguridad en la autenticación. Aunque cada servicio es libre de establecer su método de creación del valor de la firma propio, *OAuth* define tres: mediante texto plano, algoritmo RSA-SHA1 o algoritmo HMAC-SHA1.

El primero necesita de un mecanismo de capa de transporte segura, mientras que los basados en claves utilizan varios campos de la petición -URL, parámetros de *OAuth*, el cuerpo en algunos casos…- adecuadamente formateados, como base para generar la firma mediante el algoritmo, con claves secretas: clave privada en el caso de RSA, o clave compartida de las credenciales *OAuth* en el caso HMAC.

Así, el servidor de recursos podrá verificar al decodificar la firma -con las claves que solo él y los usuarios legítimos conocen-, que la petición no es fraudulenta.

Una vez descrito el formato de la autenticación, se aborda la estrategia de autorización que describe el protocolo:

Este sistema permite que el cliente sea autorizado a realizar peticiones de recursos al servidor, en nombre de su propietario; sin conocer las credenciales que este tiene compartidas con el servidor.

Para ello se establece un mecanismo de tres pasos:

1. **Petición de credenciales de usuario temporales.** El cliente obtiene del servidor unas credenciales temporales que serán empleadas para identificar la petición de acceso durante el proceso de autorización. Para ello realiza una petición autenticada -*2legged*- con un parámetro adicional: *oauth_callback*, que indique donde debe de ser redirigido el usuario tras completar el segundo paso.

2. **Autorización por parte del usuario.** El propietario del recurso es dirigido por el cliente a un *endpoint* del servidor -cuya URI contendrá el parámetro *oauth_token* con valor del identificador temporal-, para que autorice al servidor a permitir el acceso al cliente. De este modo, el cliente queda fuera del proceso momentáneamente para que el usuario pueda usar sus credenciales estándar -usuario y contraseña-. Tras ello, el usuario es redirigido por el servidor, a la dirección que fue especificada en el parámetro *oauth_callback* en el primer paso; donde recibirá un código de verificación que tendrá que proporcionar al cliente, para que complete el último paso.

3. **Petición de credenciales finales de usuario.** Una vez autorizado por el usuario, el cliente usa las credenciales temporales en una petición *3legged*, y un parámetro adicional: *oauth_verifier*, con el código de verificación recibido en el paso 2, para conseguir unas credenciales definitivas que identifiquen en adelante la autorización por parte del propietario: Tokens.

Puede apreciarse el flujo del proceso en la imagen de la Figura 2-22.

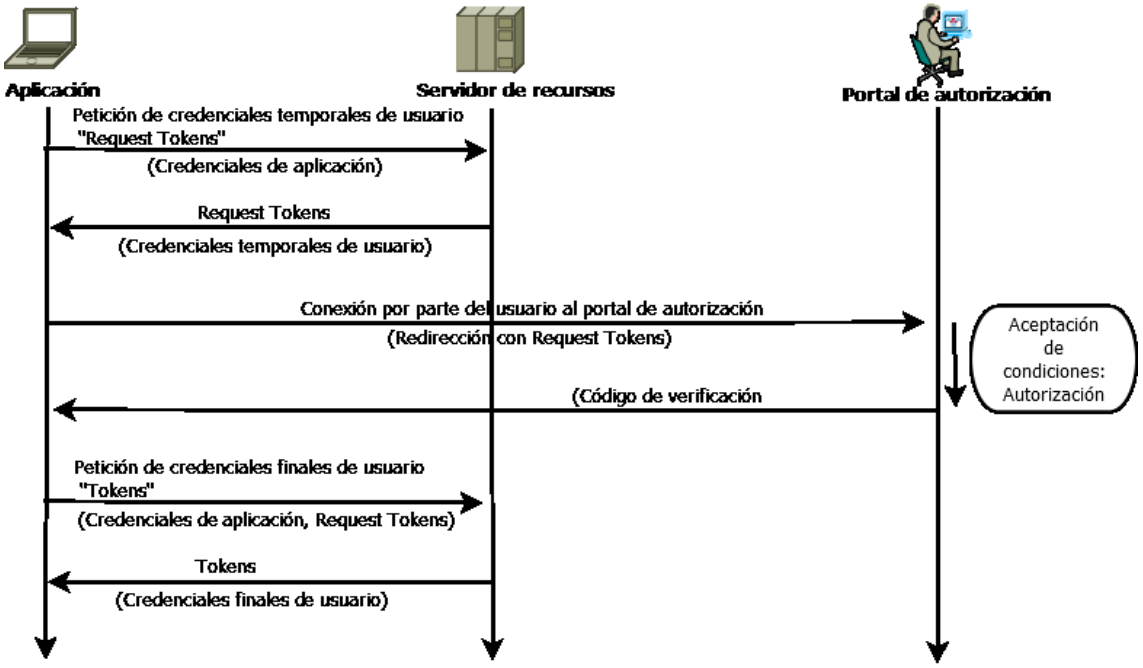


Figura 2-22 Proceso *OAuth* de autorización

Nota: *OAuth* no define como se generan y consiguen las credenciales de cliente, tampoco como se generan las de usuario; ni define como se establece la configuración adecuada de parámetros. Todo ello es dependiente de cada servicio.

2.3.1 *OAuth* para *Bluevia*

En *Bluevia* existen servicios que ofrecen información privada de usuario, y otros que ofrecen información sin propietario; por lo que se usa *OAuth* para gestionar la autorización y autenticación de aplicaciones que invoquen los servicios *3-legged* de acceso público, y la

autenticación de las peticiones a los servicios *2-legged* -tanto de acceso público, como privado-.

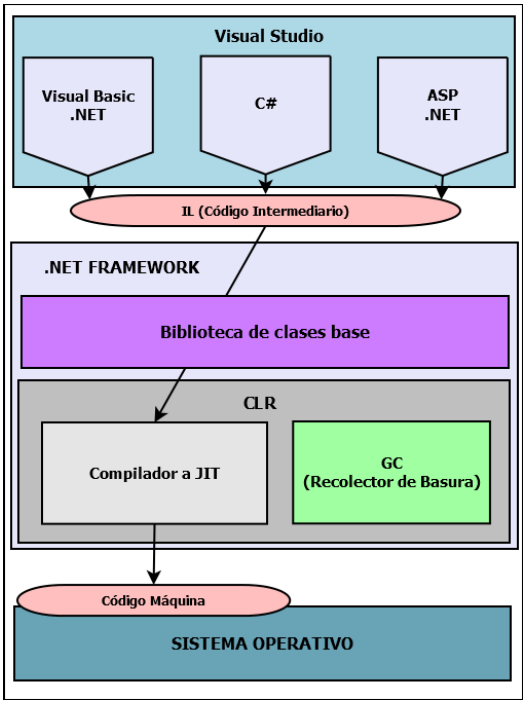
Bluevia especifica una serie de particularidades para el uso de este protocolo de autorización:

- Las credenciales de cliente son denominadas credenciales de aplicación o *ApiKeys, que* son generadas en *Portal* cuando el desarrollador da de alta una nueva aplicación.
- Dependiendo del modo de acceso operativo en el que deba funcionar la aplicación, la autorización por parte del usuario debe de realizarse en *Portal*, o *Connect.*
- Además de las posibles direcciones de *callback* definidas por *OAuth,* en *URI propia y fuera de línea* *-oob por sus siglas en Inglés-*, se añade uno extra; un MSISDN para que el usuario reciba el código verificador vía SMS.
- En el caso de los servicios de pago, los *Tokens* no identificarán la autorización dada para una aplicación completa, sino para una simple transacción; y en cada proceso de autorización relativo, deberá incluirse el parámetro especial *xoauth_apiname*.

2.4 .NET FRAMEWORK 4

.NET es un *framework* desarrollado por ***Microsoft*** con la intención de proporcionar interoperabilidad entre lenguajes y máquinas. En la Figura 2-23 se muestra un esquema de la pila de funcionamiento.

Para ello los programas escritos en cualquiera de los lenguajes soportados para ***.NET***, son compilados al mismo lenguaje intermediario -IL por sus siglas en Inglés- y ejecutados sobre el entorno CLR -*Common Language Runtime*-, en vez de directamente sobre el sistema. Este CLR, se encarga, por ejemplo, de optimizar "al vuelo" el código recibido mediante un compilador JIT -*Just In Time*-, de liberar memoria a través de un recolector de basura -GC por sus siglas en Inglés-.



1- El IDE ***Visual Studio*** permite crear programas en varios lenguajes, usando las librerías de clases base del *framework de .NET.*

2- Estos programas son compilados en un lenguaje común IL.

3- Durante la ejecución sobre el CLR, el compilador JIT, optimiza y compila los programas en código máquina, para que corran sobre el sistema operativo.

Figura 2-23 Esquema de la estructura .NET

Nota: A pesar de que .NET es un framework profundamente integrado en Windows, al estar basado en el estándar ECMA/ISO de Infraestructura de Lenguaje Común[50] -CLI por sus siglas en Inglés-; es posible la interoperabilidad entre sistemas operativos[51].

2.4.1 Algunas novedades del Framework 4

Se introducen varias mejoras en el *framework* 4, como un nuevo CLR independiente de las versiones anteriores con mejoras en la recolección de basura -permite llamar a varios recolectores de varios niveles a la vez-; un modelo de seguridad simplificado, nuevos tipos

para los lenguajes, posibilidad de ejecutar varias versiones del framework sobre un mismo servidor, aligeramiento de la versión reducida del *framework*: *Client Profile*, entre otras.

Para el desarrollo del SDK se aprovechan directamente dos de las novedades:

- **Parámetros opcionales:** Novedad para el lenguaje de ***C#,*** que permite establecer parámetros opcionales en los métodos, de modo que se elimina la necesidad de sobrecargarlos para definir distintos números de entradas.

  Estos parámetros opcionales son especificados con un valor por defecto, y después de los obligatorios.

  En los fragmentos de código de la Figura 2-24 podemos apreciar como los dos últimos parámetros de la función *InitTrusted,* de una clase padre, declaran un valor por defecto; y las invocaciones a ese método, de dos clases hijas, difieren en el número de parámetros:

Método en clase padre:

```
protected void InitUntrusted(BVMode mode
            , string consumer, string consumerSecret
            , string token = "", string tokenSecret = "")
```

Método invocado desde los constructores de las clases hijas:

```
public BV_MOSMS(BVMode mode, string consumer, string consumerSecret)
        {
            InitUntrusted(mode, consumer, consumerSecret);
        }

public BV_MTSMS(BVMode mode, string consumer, string consumerSecret, string
token, string tokenSecret)
        {
            …
            InitUntrusted(mode, consumer, consumerSecret, token, tokenSecret);
        }
```

Figura 2-24 Fragmentos de código que ilustran la nueva funcionalidad del *framework 4* para *C#*, de los parámetros opcionales.

Método a invocar:

```
public new Schemas.SimpleAdResponse GetAdvertising3L(
   string adSpace, string country = null, string adRequestId = null,
   TypeId adPresentation = 0, string[] keywords = null,
   ProtectionPolicy protectionPolicy = 0, string userAgent = "none")
```

Invocación del método:

```
adResponse = client.GetAdvertising3L(
                    protectionPolicy: ProtectionPolicy.low, //Optional
                    country: null, //Optional
                    //adRequestId: null, Optional
                    adPresentation: TypeId.image, //Optional
                    adSpace: "BV15125", //MANDATORY
                    keywords: new string[] { "Bluevia" }, //Optional
                    userAgent: "none" //Optional
                    );
```

Figura 2-25 Fragmentos de código que ilustran la nueva funcionalidad del *framework 4* para *C#*, de los parámetros nombrados.

- **Parámetros nombrados:** Otra novedad proporcionada a ***C#***, se permite la invocación de los métodos, pasando los parámetros en cualquier orden -incluso los opcionales-, siempre que vayan especificados por su nombre. Esto a priori no parece una gran ventaja para el desarrollo, pero sí lo es para el mantenimiento o ampliación de las librerías; ya que facilita la comprensión del funcionamiento de estas, y la invocación de los métodos.

  En el fragmento de código de la Figura 2-25 puede apreciarse como los parámetros nombrados pueden declararse en cualquier orden, y facilitan la rápida comprensión del código en el caso de los ejemplos del SDK 1.6 de ***.NET***:

2.4.2 Visual Studio2010

Visual Studio es el IDE creado por ***Microsoft*** para desarrollos de su *framework* en cualquier lenguaje de ***.NET.***

En él, los proyectos globales se denominan "Soluciones"; y estas pueden estar compuestas por uno o varios proyectos relacionados.

Como en los IDE más importantes, ofrece ayuda dinámica durante el desarrollo, capacidad de montar distintos tipos de servidores, funciones de depuración y compilación, entre otras.

La versión ***2010*** está enfocada al desarrollo para el *framework* 4 -puede usarse para desarrollos en frameworks anteriores, pero usa el 4 para funcionar-, y ofrece varias novedades potencialmente muy útiles, como la generación automática de diagramas de secuencia en la versión ***Ultimate***, de las que algunas que han sido usadas de forma intensiva en el desarrollo del SDK: coincidencia parcial de cadenas en la ayuda dinámica; navegación rápida entre elementos, funcionalidad para mostrar la jerarquía de llamadas de una función; y la búsqueda rápida de referencias de un término.

3 DISEÑO DE LA VERSION 1.6 DEL SDK DE ACCESO A BLUEVIA PARA *.NET*

En este bloque se abordan los aspectos relacionados con el diseño de una solución para el objetivo principal del proyecto.

- En el punto 3.1, se enumeran los requisitos existentes para la nueva versión del SDK.
- En el punto 3.2, se analizan los requisitos recopilados, para extraer las directrices generales de diseño.
- En el punto 3.3, se describe la solución alcanzada en común por el grupo de desarrollo, para el diseño una estructura general basada en las directrices extraídas en el punto anterior.
- En el punto 3.4, se profundiza en algunos aspectos de más bajo nivel, como la herencia de los módulos, o el agrupamiento de la funcionalidad.
- Finalmente en el punto 3.5, Se resume brevemente la solución diseñada.

3.1 REQUISITOS PARA EL NUEVO SDK

A continuación, se enumeran y describen los requisitos demandados por el cliente -dirección del proyecto de *Bluevia*, y dirección del grupo de SDK-, que la nueva versión del SDK debe cumplir.

3.1.1 Requisitos de alto nivel, heredados de las versiones previas del SDK

Estos requisitos, son los que el SDK cumplía en sus versiones anteriores, y atañen básicamente a la realización de un producto funcional y completo. Por lo que se mantienen para la nueva versión.

- **Ofrecer una interfaz simple en *C#*, de acceso a los servicios de *Bluevia*.**

Este es el objetivo principal del proyecto. Ya que configurar una petición contra un servicio de *Bluevia* puede resultar algo laborioso, el SDK debe facilitarle al usuario este proceso todo lo posible; de modo que se encargue de forma trasparente de empaquetar la información necesaria en el formato adecuado, gestionar la conexión de forma segura con el *endpoint,* autenticar a la aplicación y el usuario, y procesar la información recibida para devolver solo la importante. El paquete deberá de estar auditado a tiempo para la salida a producción de la plataforma.

- **Ofrecer documentación del código, una guía de uso de SDK, y código de ejemplo de uso de los servicios.**

Las clases públicas del código deberán contener la documentación necesaria para poder ser comprendidas rápidamente. Además, debe crearse una guía explicando de forma general, el funcionamiento de *Bluevia* y el uso de los diferentes servicios ofrecidos por el SDK.

Por otro lado, el paquete del SDK debe ofrecer el código de aplicaciones demostrativas funcionales, de al menos un servicio por API.

3.1.2 Requisitos adicionales de alto nivel, para la nueva versión del SDK

Estos requisitos, son los que conciernen a la nueva funcionalidad de la plataforma de *Bluevia* para su versión 1.6; y a la motivación de unificar el sistema de uso entre los SDK de los diferentes lenguajes.

- **Integrar en el proyecto, acceso para cada uno de los nuevos API privados.**

En la versión 1.6 de la plataforma de *Bluevia*, aumentan los API a los que se puede acceder por vía privada; hasta completar los servicios de *Location, Advertising, Directory,* SMS y MMS -en 1.5 sólo existía acceso privado para *Payment*-.

De modo que además de heredar los requisitos de acceso de la versión 1.5, se suma el tener que crear una interfaz simple de acceso privado a los nuevos servicios.

- **Facilitar la armonización de uso entre los SDK de los diferentes lenguajes.**

Dado que los desarrolladores que quieran usar los servicios de *Bluevia*, realizarán versiones de sus aplicaciones enfocadas a diferentes plataformas, para acceder así a una mayor cuota de mercado. Y como los plazos y presupuestos de diseño y desarrollo de esta nueva versión así lo permiten, se exige como nuevo requisito, que a partir de 1.6, una vez aprendido el uso de un SDK de *Bluevia*, resulte trivial el uso de un SDK en cualquier otro lenguaje; y de la misma forma, una vez modificado el código interno de un SDK, sea lo más sencillo posible modificar el código de otro.

3.2 ANÁLISIS DE REQUISITOS

Se analizan los requisitos enumerados en la tabla de la Figura 3-1, para extraer las primeras directrices del diseño:

1.	Ofrecer una interfaz simple en ***C#***, de acceso a los servicios de *Bluevia.*
2.	Integrar en el proyecto, acceso para cada uno de los nuevos API privados.
3.	Facilitar la armonización de uso entre los SDK de los diferentes lenguajes.
4.	Ofrecer documentación del código, una guía de uso de SDK, y código de ejemplo de uso de los servicios.

Figura 3-1 Tabla de los requisitos generales para el proyecto.

3.2.1 Análisis de los requisitos de acceso a los servicios

Comenzando con el primer y segundo requisitos -listados en la **Figura 3-1**-: para poder ofrecer al usuario del SDK una interfaz simple de acceso a los servicios que ofrece *Bluevia*, en primer lugar hay que considerar las acciones que deben realizarse para poder acceder a ellos.

Estas acciones, consisten en un proceso de conexión, autenticación y selección de opciones ofrecidas por la plataforma; siguiendo lo especificado en la web [1] para el caso de los servicios de acceso público -puede verse un resumen en el anexo 7.2-, o siguiendo las especificaciones internas a ***Telefónica I+D*** para los servicios de acceso privado.

3.2.1.1 Consideraciones sobre las especificaciones para acceder a los servicios públicos.

Las especificaciones para el acceso de los servicios públicos, dictan que:

- La conexión y comunicación, se realizan mediante HTTP sobre SSL, al *endpoint* REST o RPC -dependiendo del servicio- de servicios públicos de *Bluevia.*
- Todas las peticiones a los servicios se autentican con las credenciales de aplicación -*consumerKeys*-. Además, la mayoría de los servicios necesitan de una autorización de usuario -simbolizada mediante los *Tokens*-.

  Para ambas cosas, se usa la cabecera HTTP: "*Authorization: OAuth"* que es construida en algunos casos con modificaciones no contempladas en la RFC de dicha tecnología.

- Todos los servicios necesitan que se proporcionen ciertos datos obligatorios relativos a su funcionamiento. Además, en la mayoría de los servicios podrán existir otros datos opcionales para modificar su funcionalidad.

  Ambos tipos de datos pueden viajar en dos lugares de la petición:

  - Como parámetros en la propia URL. Si la petición es del tipo HTTP:*Get o CRUD:Retrieve*

- En el cuerpo del mensaje, en uno de los siguientes formatos: *FormURLEncoded*, XML -o JSON- y *MIMEMultipart*. Si la petición es del tipo HTTP: *Post o CRUD: Create.*

Nota: *La operación* equivalente a *CRUD:Create* en HTTP, podría considerarse *Put*; aun así, dado que las operaciones *Post* en *Bluevia*, son las que crean los recursos, se decide hacer válida la equivalencia.

- Los datos de las respuestas de muchos de los servicios, podrán variar en número y forma. Y además de poder volver empaquetados en los mismos lugares que los especificados para las peticiones, pueden volver en las cabeceras -"metainformación"- de la propia respuesta.
- De cara al SDK, el uso de RPC solo implica, con respecto a REST:
  - Cambiar parte de la URL.
  - Que para todas las peticiones, los datos viajarán en el cuerpo en formato XML.

3.2.1.2 Consideraciones sobre las especificaciones para acceder a los servicios privados.

A pesar de no formar parte del ámbito de este documento, pero dado que forma parte de los requisitos del proyecto, han de considerarse ciertas especificaciones de los servicios de acceso privado para poder explicar el diseño de forma conveniente.

Estas especificaciones, varían en algunos aspectos con respecto a las de acceso público:

- La conexión y comunicación, para todos los servicios privados, se realizan mediante HTTP sobre un modo diferente de SSL -con certificado de dos vías para autenticar a una aplicación que ya ha sido autorizada por el usuario-, y a un *endpoint* distinto del de los servicios públicos.
- La operativa es siempre bajo autenticación *2legged* -dado que no es el usuario quien autoriza a la aplicación de cara a *Bluevia*-.
- Los datos que se envíen o reciban, y los servicios de los API: pueden variar con respecto a los de acceso libre. Aun así, los datos viajan en los mismos lugares y usan los mismos formatos, que los descritos para los libres.

3.2.1.3 Enumeración de pasos abstractos a realizar, para la invocación de un servicio de *Bluevia.*

Ahora con todos esos detalles, podemos extraer los pasos genéricos que hay que seguir para invocar un servicio, tanto público, como privado. Estos pasos se ilustran en la Figura 3-2.

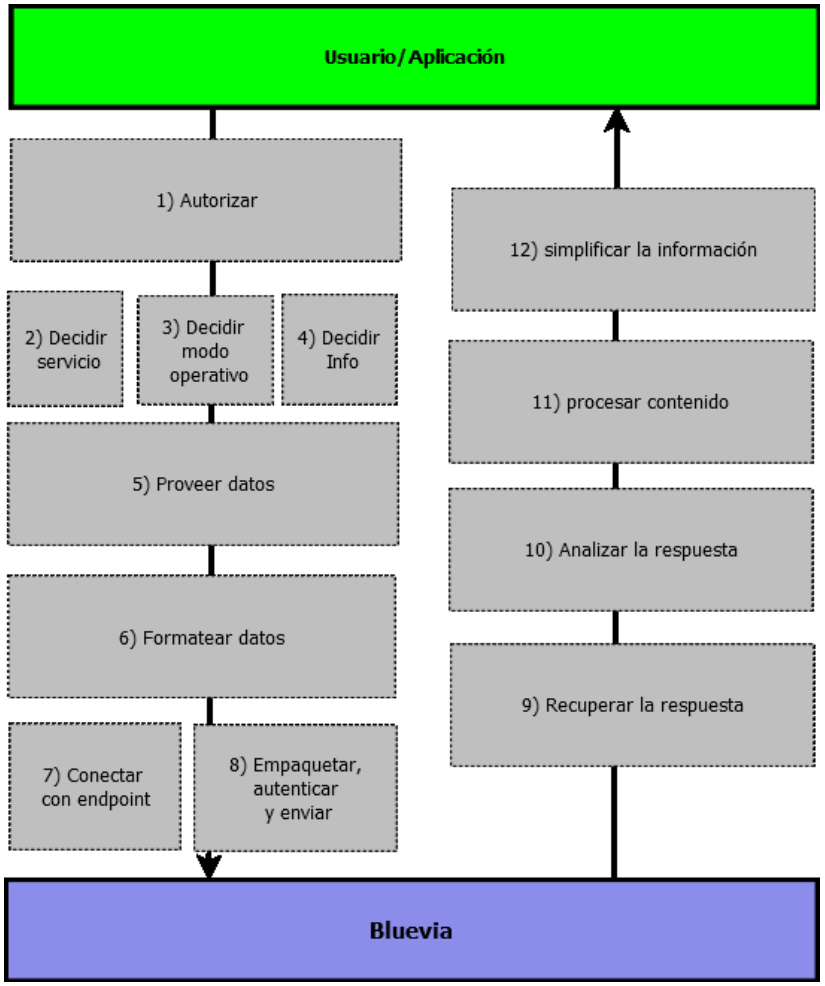


Figura 3-2 **Esquema de los pasos a realizar para invocar un servicio de *Bluevia***

Usaremos como ejemplo, el de un usuario final, de una aplicación que usa el API de *Advertising*:

Paso 1. Autorizar la aplicación (solo si es 3legged).

Para poder usar el API de *Advertising* no es necesario que el usuario autorice -de cara a ***Telefónica***- a la aplicación que esté usando las librerías, dado que el servicio es *2-3legged* -es un servicio gratuito, y no recupera información personal del usuario-.
Pero para el ejemplo, asumiremos que la aplicación hace uso de otro API que si necesite ser autorizado, por lo que el API de *Advertising para esta aplicación, tendrá que funcionar como* 3legged. Para ello, debe pasar el proceso de *OAuth*, y conseguir unos *Tokens* válidos.

Paso 2. Decidir qué servicio va a usarse.

En el caso de *Advertising* solo hay un servicio disponible, a pesar de que se pueda acceder a él de diversas formas: *simple/requests*.

Paso 3. Decidir en qué modo operativo se va a usar en el servicio.

El servicio *simple/requests*, está obligado a ser usado de forma autorizada en este caso. Por otro lado, los entornos *TEST* o *SANDBOX* están enfocados a desarrolladores de aplicaciones. Como el escenario del ejemplo es el de un usuario final, la petición será realizada contra el entorno *LIVE*, por lo que el anuncio que reciba el usuario, será real, y "monetizable".

Paso 4. Decidir qué información del servicio se quieren recuperar, o que información se quiere enviar.

De la información que puede proporcionar el servicio -anuncio en forma de imagen, o texto-, la aplicación quiere recuperar una imagen, de un anuncio para España, que haga referencia a la palabra clave "*Bluevia*".

Paso 5. Proveer los datos necesarios para que el servicio devuelva la información deseada (datos de operativa del servicio y datos exigidos por el servicio).

Los identificadores de la aplicación que accede al servicio.
Los *Tokens* que autorizan a la aplicación a realizar peticiones en nombre del usuario.
Para poder realizar la petición al servicio, se necesitará:
El modo decidido: *LIVE*.
El *endpoint* del API: https://api.bluevia.com/services/REST/Advertising
El servicio seleccionado: *simple/requests*.
La versión del servicio de *Advertising*: v1
La información deseada: imagen, para España, y referente a *Bluevia.*
Datos obligatorios: identificador de petición, espacio de anuncios, agente de usuario

Paso 6. Formatear los datos necesarios atendiendo a la especificación del servicio.

Para el caso de Advertising -como se presenta en la Figura 7-2 del anexo 7.2-, los datos de la petición van encapsulados en formato FormURLEncoded en el cuerpo. Por otro lado, habrá que generar una cabecera de Oauth adecuada a la petición.

En la Figura 3-3 vemos cómo quedaría la petición al completo:

```
POST
https://api.bluevia.com/services/REST/Advertising/simple/requests?version=v1
Authorization: OAuth
oauth_token="token",oauth_consumer_key="key",oauth_nonce="*",oauth_signature_method="HMA
C-SHA1",oauth_version="1.0",oauth_timestamp="*",oauth_signature="*"
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: api.bluevia.com
Content-Length: 158

ad_presentation=0101&ad_request_id=ad3f0f598ffbc660fbad9035122eae7404%2F10%2F2012%2017%3
A20%3A02&ad_space=BV15125&country=ESP&keywords=Bluevia&user_agent=none
```

Figura 3-3 Ejemplo de petición al API de *Advertising*

Paso 7. Conectar con el *endpoint* del servicio.

Se tendrá que abrir una vía de comunicación HTTPS contra el servidor de *Bluevia*, hacia el servicio de *simple/requests*, por la que poder enviar la petición.

Paso 8. Autenticar la petición autorizada, empaquetar, y enviar los datos de invocación del servicio.

La petición se autenticará, con la información de aplicación y de usuario de la cabecera de Oauth, y se enviará con el formato que se ha visto en el punto 6.

Paso 9. Recuperar la respuesta y cerrar el canal.

Habrá que esperar una respuesta durante un tiempo razonable, y una vez recibida, cerrar el canal si no se realizan más peticiones.

Paso 10. Analizar la respuesta y constatar que no es un error, para seguir con el proceso.

La respuesta puede ser errónea por varios motivos, información de autenticación incorrecta, información enviada o requerida al servicio incorrecta, fallo en el servidor, etc.

Paso 11. Procesar el contenido de la respuesta según el formato descrito en la especificación del servició.

En el caso de *Advertising* -como se presenta en la tabla de la Figura 7-2 del anexo 7.2-, la información devuelta tendrá que ser extraída del cuerpo de la respuesta en formato XML. En la Figura 3-4 se muestra un ejemplo de respuesta.

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=UTF-8
Content-Length: 630
Date: Thu, 04 Oct 2012 17:20:01 GMT

<?xml version="1.0" encoding="UTF-8"?>
<NS1:adResponse xmlns:NS1="http://www.telefonica.com/schemas/UNICA/REST/sgap/v1/"
id="111122010-10-18T10:00:0030c8db243ad" version="2">
<NS1:ad id="52891020" ad_placement="1" campaign="52891004" flight="52891019">
<NS1:resource ad_presentation="0101">
   <NS1:creative_element type="image">
          <NS1:attribute type="locator">
              https://api.bluevia.com/services/Ad/BlueViaBanner.png
           </NS1:attribute>
           <NS1:interaction type="click2wap">
              <NS1:attribute type="URL">
                  http://www.bluevia.com
              </NS1:attribute>
           </NS1:interaction>
    </NS1:creative_element>
</NS1:resource>
</NS1:ad>
</NS1:adResponse>
```

Figura 3-4 Ejemplo de la respuesta del API de *Advertising*

Paso 12. Simplificar la información recuperada y eliminar lo innecesario.

Una vez extraída toda la información relevante del servicio, se crea un objeto simple con los campos útiles para la aplicación -La imagen del anuncio, y la URL de la interacción-, y se devuelven a la aplicación.

Paso 13. Usar la información para lo que se considere oportuno.

3.2.1.4 Distribución de los pasos abstractos enumerados, entre el usuario/aplicación y el SDK

Una vez desglosado el proceso de realizar una petición a un servicio de *Bluevia*, y atendiendo a ofrecer un acceso simplificado a dichos servicios como requisito principal. Se dejarán, para ser realizados por el usuario del SDK, tan solo los pasos mínimos indispensables, encargándose el SDK del resto del trabajo, como se puede ver en la Figura 3-5.

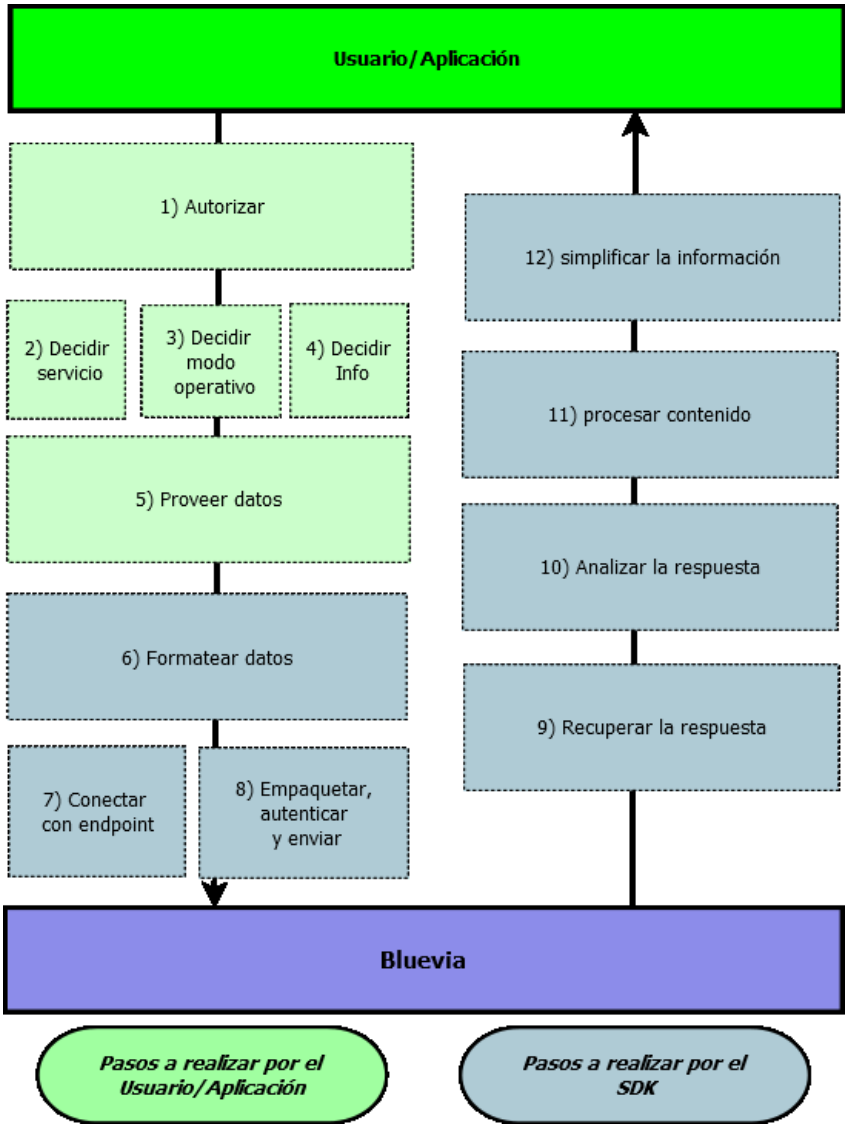


Figura 3-5 **Distribución del trabajo para el usuario y el SDK, en los pasos a realizar para invocar un servicio de *Bluevia***

Los pasos que deberán realizar usuario/aplicación son :

Paso 1. Autorizar la aplicación (solo si es 3legged).

Paso 2. Decidir qué servicio va a usarse.

Paso 3. Decidir en qué modo operativo se va a usar en el servicio.

Paso 4. Decidir qué información del servicio se quieren recuperar, o que información se quiere enviar.

Paso 5. Proveer los datos necesarios para que el servicio devuelva la información deseada. Pero en este caso, tan solo tendrá que proveer los datos que sean estrictamente dependientes del usuario final

Paso 13. Usar la información para lo que se considere oportuno.

Los pasos que deberá de realizar el SDK:

Paso 1. Autorizar la aplicación (solo si es 3legged).

Será el usuario final quien tenga que autorizar realmente a la aplicación completando el proceso de OAuth, pero el SDK puede ayudarle configurando y realizando las peticiones de autorización.

Paso 5. Proveer los datos necesarios para que el servicio devuelva la información deseada.

El SDK se puede encargar de completar los datos necesarios para invocar servicios, asignándoles un valor por defecto, según el usuario haya decidido la operativa el servicio etc…

Paso 6. Formatear los datos necesarios atendiendo a la especificación del servicio.

Paso 7. Conectar con el *endpoint* del servicio.

Paso 8. Autenticar la petición autorizada, empaquetar y enviar los datos de invocación del servicio.

Paso 9. Recuperar la respuesta y cerrar el canal.

Paso 10. Analizar la respuesta y constatar que no es un error, para seguir con el proceso.

Paso 11. Procesar el contenido de la respuesta según el formato descrito en la especificación del servició.

Paso 12. Simplificar la información recuperada y eliminar lo innecesario.

3.2.2 Análisis del requisito sobre servicios privados

El que aumenten los API de acceso privado, hasta casi igualar los de acceso público, y haya que ofrecer acceso a ellos, implica:

Un uso más intenso de la conexión con SSL de dos vías, que el que se daba en las versiones anteriores. Por lo que este aspecto, no debe quedar como algo marginal dentro del SDK.

- La gestión de SSL, debe de atenderse como una opción principal dentro de la estructura.

Servicios y parámetros adicionales o diferentes a los especificados para el acceso público.

- Habrá que mantener el acceso a los servicios privados, separado de la versión pública del SDK, sin que ello afecte a futuros desarrollos.

3.2.3 Análisis del requisito de armonización entre lenguajes

Ya se ha visto en el punto 3.2.1.3, que el trabajo por parte del usuario para poder acceder a un servicio, se reduce a un desglose en 5 pasos. Por lo que para poder armonizar el uso entre los SDK de los diferentes lenguajes, lo más apropiado es ofrecerle la misma interfaz en todos los SDK para realizar ese trabajo.

Así, autorizar una aplicación, seleccionar los servicios, los modos operativos, proveer o recuperar los datos entre otras acciones, se hará de la misma forma -aunque sea de manera abstracta- que en el resto de SDK, dentro de lo posible.

- De cara a la interfaz de uso, mismos nombres de clientes y métodos, mismos parámetros y mismos objetos devueltos.

Por otro lado, para cumplir con la segunda parte del requisito, la de poder modificar el código interno de los SDK de la misma forma, y teniendo en cuenta que las diferencias entre los lenguajes no son meramente sintácticas -lo que haría imposible escribir el mismo código o estructuras de código- , se decide replicar en todos los SDK, al menos la misma estructura de alto nivel, siempre que sea posible.

Así, una estructura general común, y una interfaz de uso similar para los diferentes SDK –sino las mismas, dentro de lo posible-, se hacen necesarias para cumplir con ello.

- Misma estructura de niveles y mismas clases principales de la estructura.

3.2.4 Análisis del requisito de documentación y demos

Sobre este punto, se seguirá con la tónica de la anterior versión del SDK, de generar un documento a partir de comentarios en el código, y actualizar las guías de uso a las novedades aportadas por la plataforma. Mientras que se creará un proyecto de aplicaciones demostrativas que ejecute los servicios disponibles en al menos una de sus formas posibles.

Este requisito no se considerará para el diseño de la librería del SDK.

3.2.5 Análisis de la experiencia con las versiones anteriores del SDK.

Aprovechando la experiencia previa en el mantenimiento de las versiones anteriores del SDK, se pueden extraer varias directrices de diseño adicionales, a las ya concluidas por los requisitos exigidos.

Estas ideas surgen para evitar las limitaciones que aparecían durante el desarrollo sobre el diseño de las versiones anteriores -punto 2.1.7-, y poder ofrecer así, un código más limpio, y fácil de encarar.

Una de las mayores dificultades a la hora de ampliar el SDK, era la de cubrir la funcionalidad demandada sobre OAuth, el empaquetamiento de cierto tipo de datos, o la gestión de conexiones, usando *DotNetOpenAuth*. Pues *Bluevia* -en algunos de sus servicios- demandaba algunas particularidades que no estaban soportadas por dicha librería. Obligando esto a saltarse el flujo de ejecución general y replicar código que cubriese tales carencias.

Así, a diferencia de los SDK de otros lenguajes -que mantendrían la dependencia con librerías externas para gestionar estos, u otros aspectos-, aprovechando el potencial del Framework de ***.NET***, y apoyado en la experiencia de haber desarrollado el código que cubría las carencias de la librería, se presenta la posibilidad de eliminar la dependencia con *DotNetOpenAuth.*

- Eliminar cualquier dependencia de librerías ajenas al Framework de ***.NET***

Otro de los problemas que acababan surgiendo a la hora de modificar las versiones anteriores del SDK, era lo interdependientes que resultaban las clases entre si, y lo grandes y difíciles de modificar que acababan siendo algunas.

Por lo que al ser este proyecto una refactorización total, se presenta la oportunidad de realizar un diseño que evite, dentro de lo posible, este problema. Distribuyendo la funcionalidad específica en pequeñas clases o módulos independientes -y llegado el caso fácilmente sustituibles.

- Distribuir la funcionalidad en módulos manejables, lo más herméticos posibles.

3.3 DISEÑO DE MEDIO NIVEL

Una vez extraídas las directrices generales de diseño, enumeradas en la Figura 3-6, se perfila la estructura del proyecto.

1.	Pasos del punto 3.2.1.4, a realizar por el usuario/aplicación, para invocar un servicio: 1, 2, 3, 4, 5, 13
2.	Pasos del punto 3.2.1.4, a realizar por el SDK, para invocar un servicio: 1, 5, 6, 7, 8, 9, 10, 11, 12
3.	Misma estructura de niveles y mismas clases principales de la estructura para los SDK de diferentes lenguajes.
4.	Distribuir la funcionalidad en módulos manejables, lo más herméticos posibles.
5.	De cara a la interfaz de uso, mismos nombres de clientes y métodos, mismos parámetros, mismos objetos devueltos …
6.	La gestión de SSL, debe de atenderse como una opción principal dentro de la estructura.
7.	Habrá que mantener el acceso a los servicios privados, separado de la versión pública del SDK, sin que ello afecte a futuros desarrollos.
8.	Eliminar cualquier dependencia de librerías ajenas al Framework de ***.NET***

Figura 3-6 Tabla de las directrices generales de diseño para el proyecto.

Atendiendo a la distribución de pasos abstractos definidos en el punto 3.2.1.4, que hay que realizar para poder invocar un servicio; la estructura del proyecto del SDK debe enfocarse en resolver los pasos que designa la segunda directriz: 1, 5, 6, 7, 8, 9, 10, 11, 12, y ofrecer una interfaz de uso de cara al Usuario/Aplicación que permita resolver de forma adecuada los pasos de la primera: 1, 2, 3, 4.

Para cumplir con ello, y desde una perspectiva general, el SDK se divide en tres niveles -o capas- de funcionalidad diferenciada. Cada uno de ellos ofreciendo al nivel superior una interfaz de configuración y uso (ver **Figura 3-7**):

- **Un nivel de acceso a la red.**

Con el objetivo de establecer conexiones seguras de red, autenticar las sesiones, así como enviar y recuperar los mensajes.

Este nivel gestionará los pasos abstractos: 7, 8, 9, 10, de los definidos en el punto 3.2.1.4.

- **Un nivel de inteligencia común.**

"Serializará" y "Parseará" el contenido de los objetos encapsulados en los mensajes de red (acorde a los objetos interfaz especificados por *Bluevia*), usa el nivel inferior de paso de mensajes.

Este nivel gestionará los pasos abstractos: 6, 11, de los definidos en el punto 3.2.1.4.

- **Un nivel de inteligencia específica de API.**

Es la interfaz de entrada de la librería, contiene la lógica relativa a los servicios específicos de cada API -por lo tanto, radica aquí la separación entre servicios públicos y privados-, y la configuración y uso del nivel inferior de inteligencia común.

Este nivel gestionará los pasos abstractos: 5, 12, de los definidos en el punto 3.2.1.4.

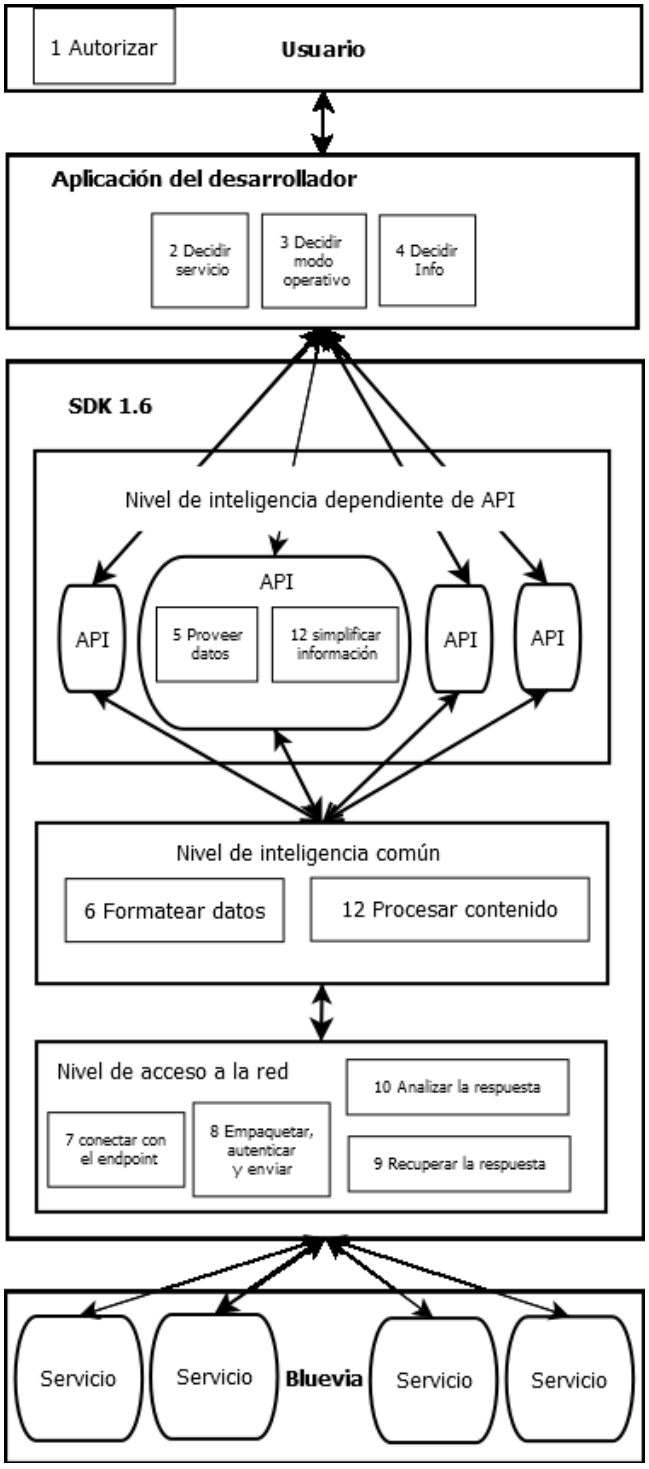


Figura 3-7 Estructura general del SDK versión 1.6

Así, con esta distribución en niveles, se atiende en parte a las directrices 4 y 5, separando las distintas funcionalidades particulares del proyecto, en módulos lo más cerrados posibles, para poder modificarlas fácilmente en caso de ampliación o cambio de especificaciones, y sentando las bases de una estructura común a la de los SDK del resto de lenguajes.

3.3.1 Nivel de acceso a la red

La función de este nivel es la de establecer, autenticar y mantener la comunicación con la interfaz de red de los servicios ofrecida por *Bluevia*.

En él, se cubren las directrices de diseño referentes la conexión de red:

Directriz 1. Las relativas a los pasos abstractos: 7, 8, 9, 10, de los definidos en el punto 3.2.1.4.

Directriz 2. La gestión de la conexión a servicios privados mediante SSL de 2 vías.

Directriz 3. La eliminación de la dependencia de librerías ajenas, que en versiones anteriores gestionaban las conexiones, la autenticación.

Directriz 4. Atender en lo posible a la modularidad hermética.

Para el diseño de este nivel, además de las 2 primeras directrices recién listadas, y las especificaciones de acceso a los servicios públicos existentes -consideradas en el apartado 3.2.1.1-, se atiende en particular a la cuarta directriz, para separar la interdependencia existente entre las tecnologías necesarias, de modo que se evite una integración monolítica al estilo de la versión 1.5 del SDK (ver: Limitaciones de las versiones anteriores del SDK para ***.NET***). Y facilitar así los posibles cambios de especificaciones que puedan surgir en un futuro –por ejemplo: Autorización *OAuth2*.0, en vez de *OAuth*-.

También, considerando la tercera directriz, se diseñan al completo módulos que gestionen los procesos de autenticación, empaquetamiento y conexión -envío y recuperación-, para cubrir la ausencia de la librería externa *DOTNetOpenAuth*.

- Se establece un elemento **Conector**, como gestor principal del nivel. Englobando toda la funcionalidad requerida a este y ofreciendo así una interfaz única al nivel superior.

- El protocolo de intercambio de mensajes es el de HTTP sobre SSL. Este protocolo aúna formatos de empaquetamiento y gestión de la conexión, por lo que ambas funcionalidades se llevarán a cabo en un módulo específico para ello. **Módulo de HTTPS**.

  Será en este módulo donde se efectúe finalmente la conexión segura usando uno de los dos modos SSL ofrecidos. El simple para los servicios públicos, y del de dos vías para los servicios privados.

- A pesar de que para el caso de *OAuth,* la autenticación va empotrada -algo por encima- en HTTPS. No tiene por qué ser así en todos los sistemas de autenticación. Por lo que, considerando los posibles cambios futuros, se decide que esta funcionalidad pertenezca a un módulo independiente. **Módulo de autenticación**.

- Aunque se realizasen cambios de tecnología en las especificaciones, el paradigma de operaciones CRUD (*Create, Retrieve, Update & Delete*) probablemente seguiría sirviendo para el cometido de definir y configurar -a varios niveles- las acciones de invocación de un servicio. Además tiene una traducción prácticamente directa con las operaciones HTTP demandadas por dichos servicios de *Bluevia*. Por lo que se selecciona este, como interfaz estándar.

Nota: En este interfaz, se expondrá la operación *Update* sin implementar, ya que no es usada por servicio alguno, pero puede serlo en un futuro.

Así, atendiendo a los puntos anteriores, se establece un diseño del nivel de acceso a la red,, como puede verse en la Figura 3-8Detalle del Nivel de paso de mensajes, compuesto por un **Conector** que ofrezca la interfaz CRUD hacia el nivel superior, con un **Módulo de HTTPS**, y otro **Módulo de Autenticación**.

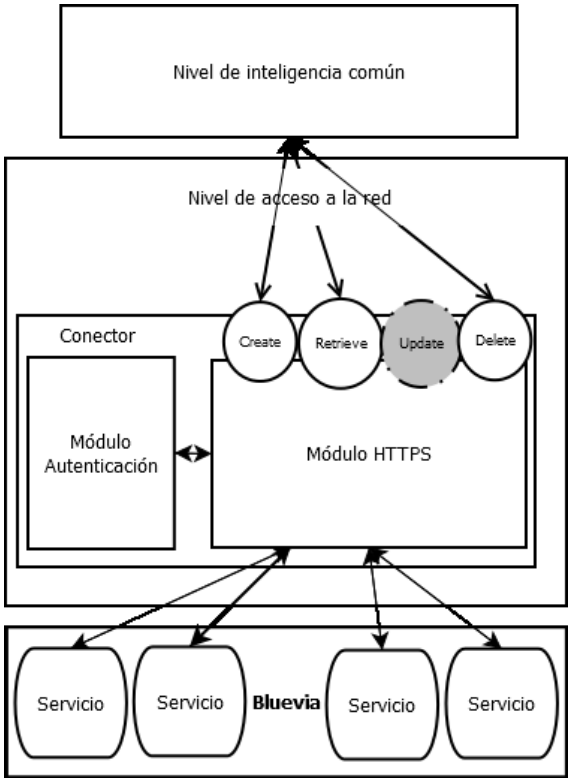


Figura 3-8 Detalle del Nivel de paso de mensajes

3.3.1.1 Conector

Este **Conector** es independiente de la tecnología de acceso a los servicios. Se diseña para que básicamente atienda solo a las necesidades abstractas de la plataforma y englobe al resto de la funcionalidad del nivel, delegando en los módulos de **Autenticación** y **HTTPS** la gestión de la conexión, y la transmisión y recepción de mensajes.

Así, contempla en su construcción a los elementos básicos de *Bluevia*: los identificadores de aplicación y de usuario -tanto en *Tokens* como en certificado de 2 vías-. Y ofrece funcionalidad para adecuar los módulos delegados, a las particularidades de los servicios que se deseen invocar.

3.3.1.2 Módulo HTTPS

Este módulo representa “la base del pilar principal” del proyecto, por el que se empaquetarán, enviarán y recuperarán, todas las peticiones a los servicios. En definitiva es un cliente simple de HTTPS.

Este módulo, expone la interfaz CRUD final, desarrollando el flujo mostrado en la Figura 3-9, de cada operación ofrecida:

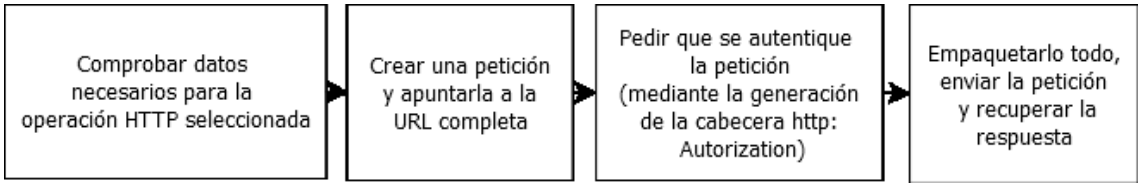


Figura 3-9 **Flujo de la funcionalidad de los métodos CRUD en el módulo HTTPS**

Pudiéndose descomponer el último bloque en el flujo de funcionamiento de la Figura 3-10.

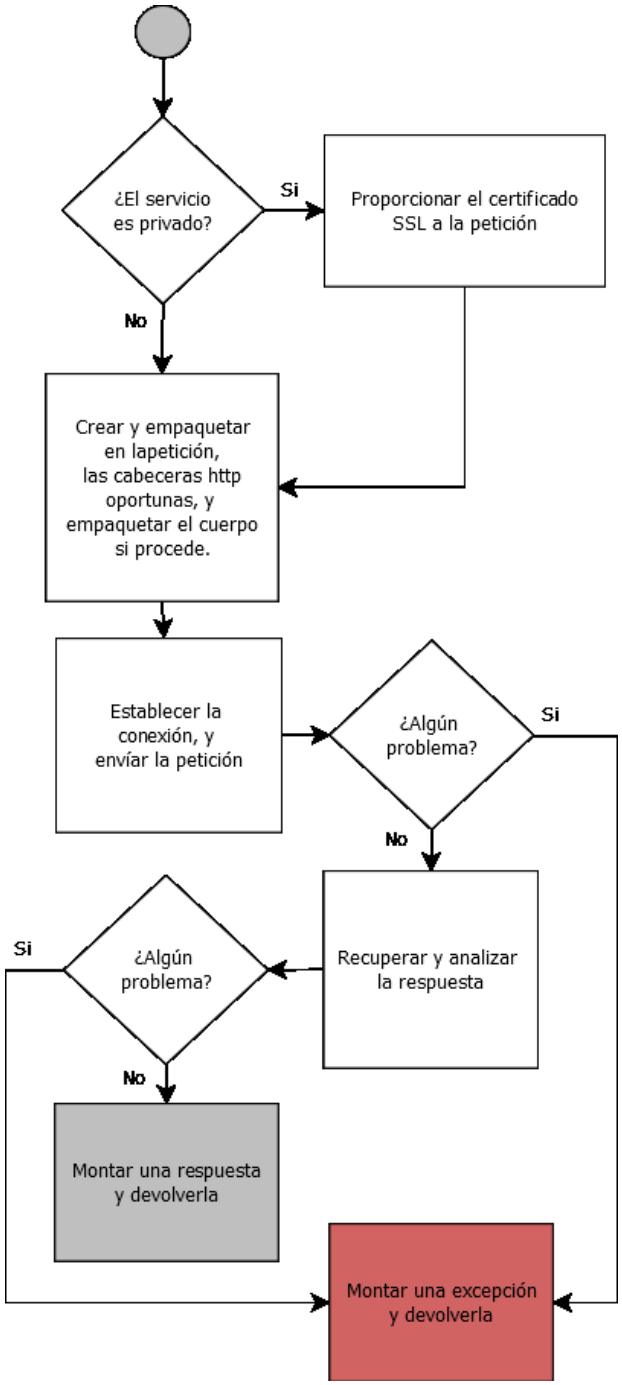


Figura 3-10 **Flujo de montaje, envío y recuperación de una petición http/*Bluevia***

a. Respuestas

Para no complicar el manejo del conector, los métodos de entrada CRUD devolverán siempre el mismo objeto. Esto es, una Respuesta genérica, con los campos necesarios para contener toda la información recuperada de cualquier servicio de *Bluevia*. Mostrada en la Figura 3-11.

- Un campo de código de estado de la respuesta.
- Un campo para un mensaje textual de la respuesta (o para completar el estado, con una descripción de este).
- Un campo de "metainformación" de la respuesta (o información adicional).

Que pueda ser procesada adecuadamente en los niveles superiores que contienen la inteligencia necesaria para ello.

- Un campo binario para el contenido del mensaje -o cuerpo- de respuesta.

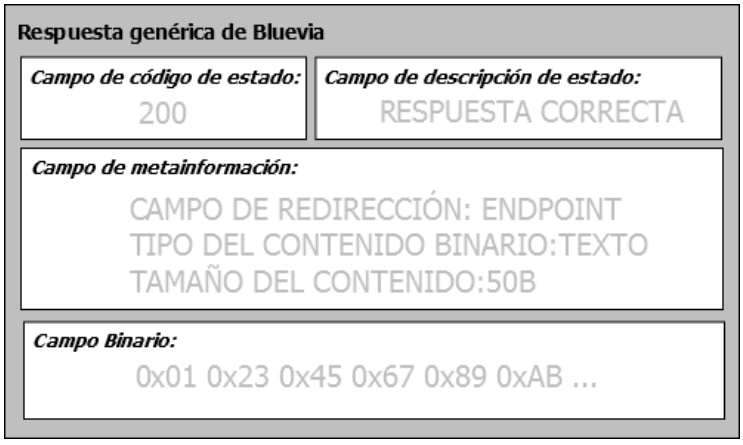


Figura 3-11 Representación gráfica de una respuesta genérica de *Bluevia.*

Este tipo de respuesta, conlleva un incremento en el coste de proceso y memoria, que puede resultar innecesario en la mayoría de los servicios que serán usados -muchos de ellos no contendrán un cuerpo-. Pero resulta un diseño mucho más limpio y ordenado, y facilita la labor de hacer el nivel modular.

b. Errores de red

Este nivel aparecerán las excepciones más importantes. Aunque los controles de parámetros hace poco probable que surja algún problema con los datos que se reciban desde la interfaz, existen focos a nivel de red que arrojarán excepciones que deban ser controladas:

- Falta o pérdida de la conexión de red.
- Incapacidad de conectar con el *endpoint.*
- Servidores que no contesten o lo hagan de forma inadecuada.
- Respuestas de error, por culpa de peticiones con datos incorrectos.

Los tres primeros tipos de excepciones se modelan con una **Excepción genérica**, con un código de error y un mensaje que describa el origen del problema.

Pero para el último tipo, se necesitan los mismos campos que para las respuestas. Ya que, aunque erróneas, son respuestas completas con "metainformación" y un cuerpo describiendo cuales de los datos enviados en la petición, provocaron el problema en el servicio. Este tipo será modelado con la **Excepción particular**.

3.3.1.3 Módulo de Autenticación

No todas las peticiones http tienen que ser autenticadas. Pero los servicios de *Bluevia* sí. Por ello se separa esta funcionalidad, de la del módulo de HTTPS.

Así, se establece en el nivel de acceso a red este módulo, para autenticar las peticiones mediante la generación según la RFC de *OAuth* [11] -con detalles extra por parte de la plataforma- del contenido de la cabecera *Authorization*. Este contenido consiste en una lista de pares "Parámetro *OAuth*-Valor", precedida por la clave: *"OAuth"*, como se pudo ver en el punto 2.3.1: *OAuth* para *Bluevia*.

En definitiva, este módulo recibirá del **Conector** la información necesaria sobre la petición a realizar, y ejecuta los pasos descritos por las tecnologías -*OAuth* y *Bluevia*-, para generar el contenido de los campos de la cabecera.

3.3.2 Nivel de inteligencia común

Este nivel engloba las herramientas necesarias para resolver las necesidades del acceso a *Bluevia*. Es decir, no dicta los requisitos específicos de cada servicio -tarea que realiza el nivel de inteligencia específica de API- , sino que los ejecuta.

En él, se cubren las directrices de diseño referentes al formateo de la información:

Directriz 1. Pasos a realizar por el SDK para invocar un servicio: 6, 11, de los definidos en el punto 3.2.1.4.

Considerando también:

Directriz 2. Eliminar cualquier dependencia de librerías ajenas al Framework de ***.NET***.

Directriz 3. Distribuir la funcionalidad en módulos manejables, lo más herméticos posibles.

Para cumplir con el objetivo del nivel, se diseñan los siguientes elementos:

- Se establece un **Cliente básico**, que gestionará el nivel inferior de acceso a la red -a través de la interfaz de uso CRUD que en él se describió-, y las tareas de formateado y procesado de la información que demande el nivel de inteligencia específica de API.
- Dado que la funcionalidad relativa al formato de la información, es bastante extensa; y que se ha decidido eliminar la librería ***DOTNetOpenAuth*** -que en versiones anteriores

del SDK realizaba parte de este cometido-; se crearán módulos independientes enfocados a resolver un formato en concreto. "**Serializadores**" para formatear la información, y "**Parseadores**" para procesarla.

Esto cubre la siguiente funcionalidad:

- Exponer las operaciones CRUD como parte de la interfaz para usar este nivel. Ya que además de cubrir las necesidades de los servicios, ofrecen cierta linealidad con la interfaz de uso del nivel inferior, facilitando así el desarrollo del proyecto.
- Establecer en este nivel funcionalidad que permita configurar la operativa de acceso de los servicios. Por ejemplo: Definir si los API que se usarán serán *Trusted*, o si los servicios empleados no van a usar autenticación de usuario.

Así, atendiendo a estos puntos, se establece un diseño del nivel de inteligencia común, compuesto, como puede verse en la Figura 3-12, por un **Cliente básico** que ofrece la interfaz CRUD, y un sistema de configuración del modo operativo hacia el nivel superior; gestiona al nivel inferior; y ejecuta los "**Serializadores**" y "**Parseadores**".

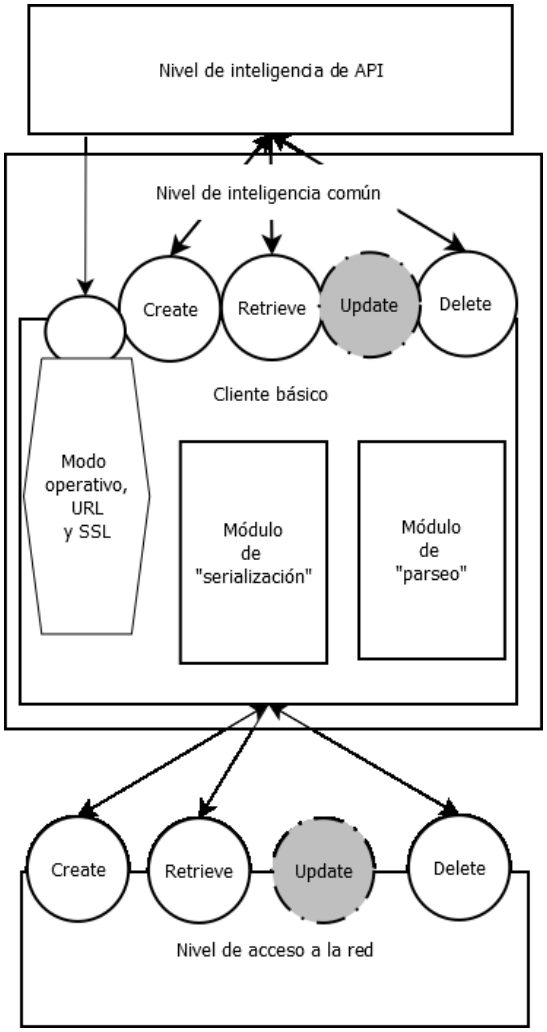


Figura 3-12 Detalle del nivel de inteligencia común

3.3.2.1 Cliente básico

Dado que esta pieza será la encargada de usar al **Conector** de acceso a la red contiene los elementos necesarios para configurarlo y manejarlo:

- Un campo y funcionalidad para construir y almacenar la URL, del *endpoint* referente al API del servicio seleccionado.
- Un campo y funcionalidad para almacenar y emplear el modo operativo seleccionado.
- Funcionalidad para montar el certificado de seguridad para SSL de dos vías.
- Exposición y Desarrollo del control de los métodos CRUD del nivel de acceso a la red, mostrado en el flujo de la Figura 3-13.

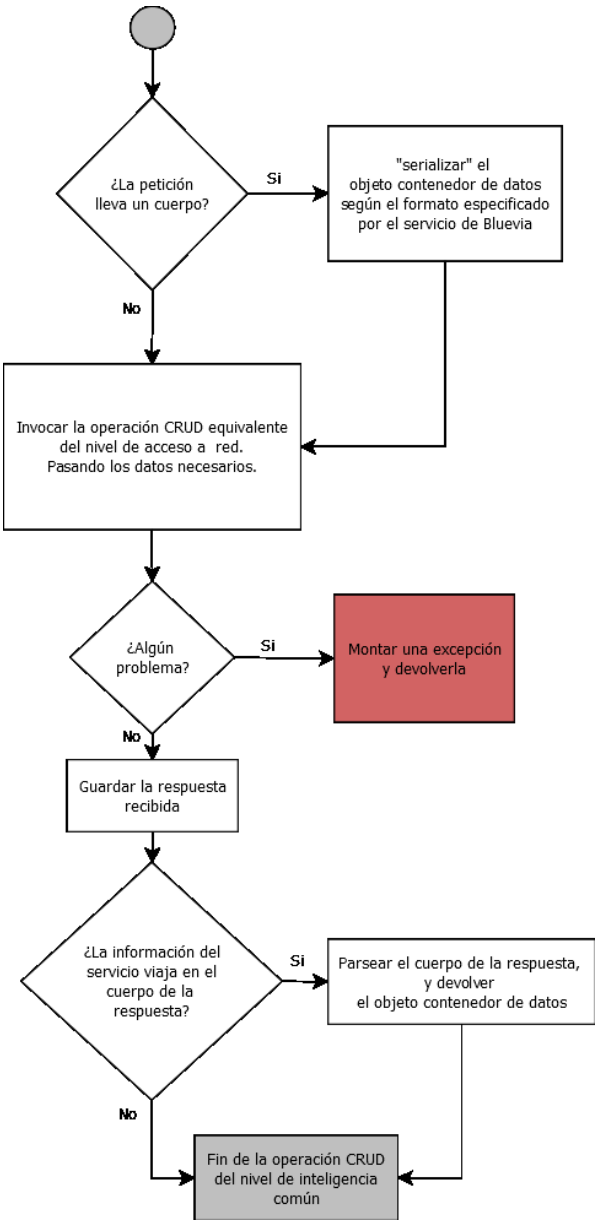


Figura 3-13 **Flujo de la funcionalidad de los métodos CRUD en el cliente básico**

3.3.2.2 "Serializadores"

Estos elementos estarán especializados en transformar a un formato en concreto -*FormURLEncoded*, XML/JSON o *MIMEMultipart*-, cualquiera de los objetos contenedores de datos definidos por *Bluevia* para invocar los servicios -ver anexo 7.2-. De modo, que el resultado de dicha operación pueda ser pasado directamente al nivel inferior para que lo inyecte de forma genérica en el cuerpo de la petición.

Así, existirá un módulo por cada uno de los tres formatos necesarios. Y dado que entre XML y JSON, XML resulta más compatible con la plataforma en el momento del desarrollo de este proyecto, tan solo se implementará esta opción en el SDK -dejando para futuros desarrollos, la posibilidad de implementar también, o de forma única JSON-.

Dado que, tanto para el formateo en *XML* como para el de *MIMEMultipart*, especificados por *Bluevia*, existen requisitos particulares solo de dicha plataforma, que harían incompatible el uso de los "serializadores" contra otras plataformas. Se subdividirán los módulos de formateo -atendiendo a la directriz de hermetizar la funcionalidad-, en uno dedicado a cubrir la tecnología de forma estándar, y otro dedicado a aportarle la parte específica de *Bluevia.*

Todos estos "**Serializadores**" tendrán una interfaz común, que permita usarlos de la misma forma, como se muestra en la Figura 3-14.

- Una función, con el objeto a "serializar" como entrada, y una salida plana, que pueda ser inyectada directamente en el cuerpo de una petición.

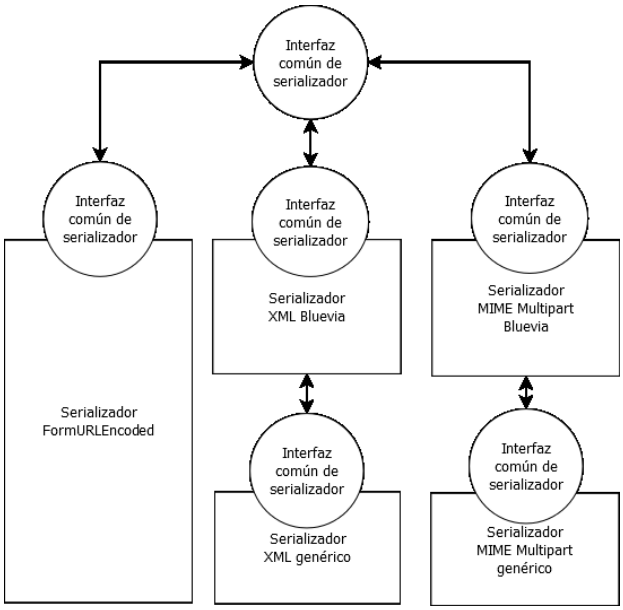


Figura 3-14 Descomposición del módulo de "Serializadores"

3.3.2.3 "Parseadores"

Estos elementos funcionarán de forma inversa "**Serializadores**", para procesar, según uno de los formatos disponibles -*FormURLEncoded*, XML/JSON o *MIMEMultipart*-, el cuerpo de las respuestas recibidas, y transformarlo de forma automática en un objeto contenedor de datos de los especificados por *Bluevia* (ver anexo 7.2).

Por el mismo motivo que en el caso de los "**Serializadores**", existirá un módulo por cada uno de los tres formatos necesarios, y se ignorará JSON. Pero a diferencia de estos, la funcionalidad de procesado extra, enfocada a las particularidades de la plataforma, no causaría problemas a la hora de procesar mensajes estándar -estos mensajes simplemente no activarían la funcionalidad de procesado extra-, por lo que no hay necesidad de realizar subdivisión de módulos.

Todos estos "**Parseadores**" tendrán también una interfaz común, que permita usarlos de la misma forma, como se muestra en la Figura 3-15.

- Una función, con el cuerpo plano a procesar como entrada, y un objeto contenedor de datos completo, cuyos campos puedan ser accedidos inmediatamente, por el nivel de inteligencia de API.

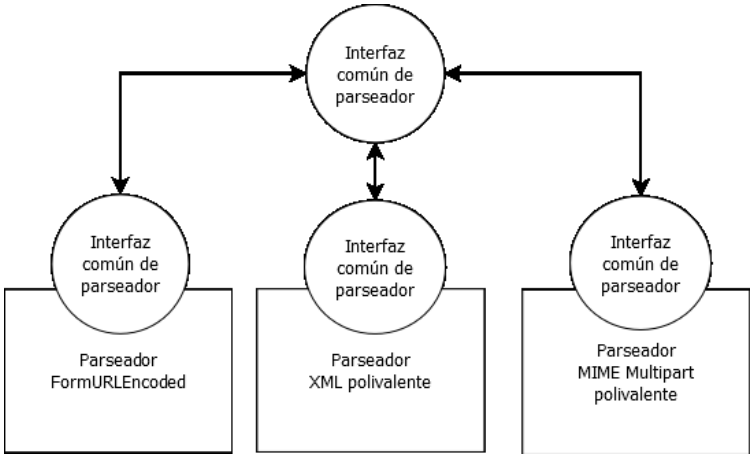


Figura 3-15 Descomposición del módulo de "Parseadores"

3.3.3 Nivel de inteligencia de API

Una vez diseñadas las herramientas que podrán ejecutar el acceso a los servicios de *Bluevia*, es el momento de diseñar el nivel donde radican: la inteligencia para usar tales herramientas de forma adecuada, y la interfaz de cara al usuario.

Las directrices que debe cubrir este nivel son:

Directriz 1. Pasos a realizar por el usuario/aplicación, para invocar un servicio: 1, 2, 3, 4, 5, 13, de los definidos en el punto 3.2.1.4.

Directriz 2. Pasos a realizar por el SDK, para invocar un servicio: 5, 12, de los definidos en el punto 3.2.1.4.

Directriz 3. De cara a la interfaz de uso, mismos nombres de clientes y métodos, mismos parámetros, mismos objetos devueltos, que el resto de SDK.

Directriz 4. Habrá que mantener el acceso a los servicios privados, separado de la versión pública del SDK, sin que ello afecte a futuros desarrollos.

Directriz 5. Distribuir la funcionalidad en módulos manejables, lo más herméticos posibles.

Así, atendiendo a la directriz 2, este nivel engloba los flujos de manejo de todos los servicios agrupados por API, y expone una interfaz que contemple los requisitos 1 y 3 para cada uno de ellos.

Como *Bluevia* está diseñada para que los API sean bloques de servicios que comparten lógica e información -objetos, modos de acceso, *endpoints*-, se aprovecha esta característica para ofrecer solo un punto de entrada a esta parte común, estableciendo un módulo por API existente.

Este módulo, mostrado en la Figura 3-16, expondrá un punto de entrada por servicio existente; que ofrezca al usuario la posibilidad seleccionar, o aportar, la información necesaria que de él dependa; y que desarrollara el flujo del manejo de las herramientas disponibles para obtener la respuesta deseada.

- Como se tiene que atender a la directriz de separar el acceso público del privado, este módulo de API se subdividirá en dos partes:
    - Un **Cliente común**, que aúne la funcionalidad común a los dos accesos al API.
    - Dos **Clientes finales**: Uno público y otro privado. Que solo aportarán la funcionalidad específica de cada tipo de acceso. Dicha funcionalidad consiste en los datos de autenticación -*Tokens*, para algunos casos de cliente público; o certificado SSL, para todos los clientes privados-, y opcionalmente, campos de datos para un servicio, o servicios al completo.

      Serán estos clientes los que se ofrezcan finalmente como interfaz de uso al exterior, para utilizar el SDK al completo.

  En el proyecto que nos atañe, solo se entregará el cliente final público.

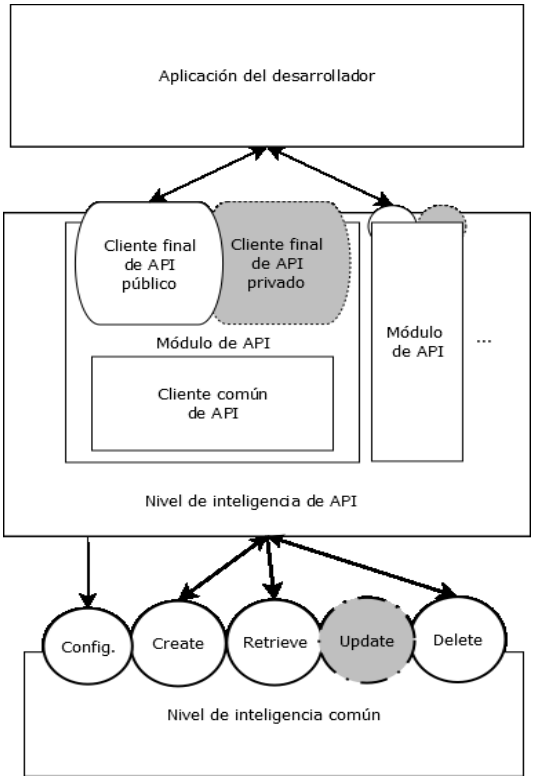


Figura 3-16 Detalle del nivel de inteligencia de API

3.3.3.1 Cliente común

Este elemento ilustrado en la Figura 3-17, concentra la inteligencia de API, común a los dos tipos de accesos, desarrollando la lógica del manejo de las herramientas para ejecutar la invocación del servicio, y la de recuperación de datos posterior.

Así, seleccionará los "**Parseadores**" y "**Serializadores**" que necesiten sus servicios, y se encargará de proporcionar a los niveles inferiores tanto el modo operativo, como los datos de autorización y autenticación provistos por el usuario del SDK.

Además, por cada servicio que defina el API, se define al menos un módulo base que desarrolle la lógica correspondiente -"*servicioProcess*"-. Aunque si el servicio definido ofrece varias opciones de funcionamiento -por ejemplo *2/3legged*-, se crearán más módulos extra que aporten la lógica relativa.

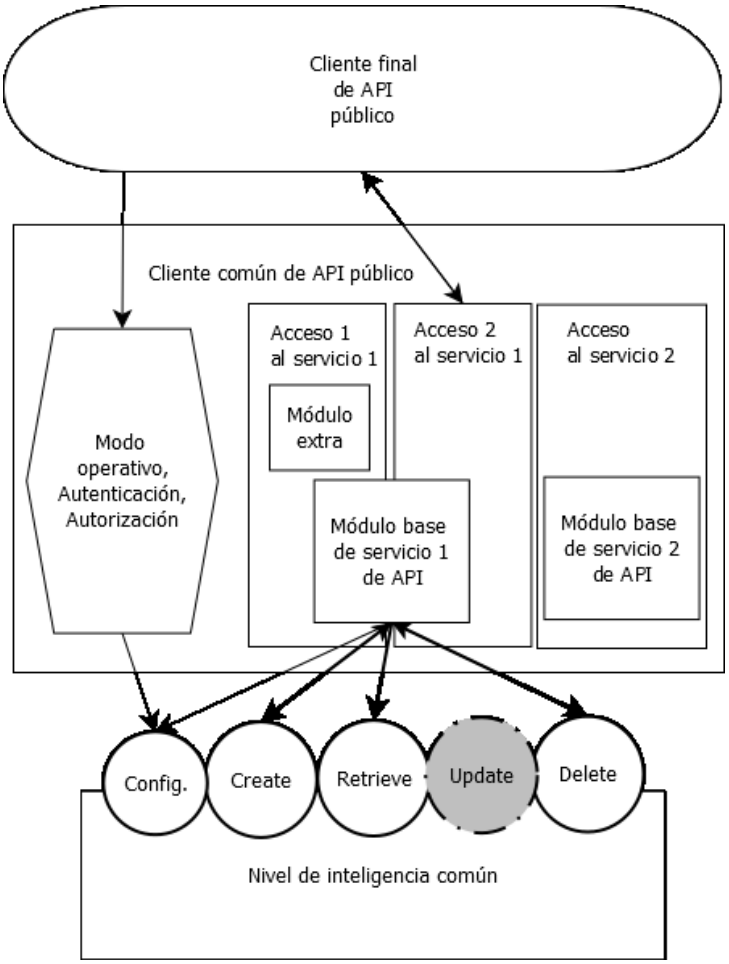


Figura 3-17 Detalle de la estructura del cliente común de un API.

La funcionalidad de estos módulos se representa en el flujo de la Figura 3-18.

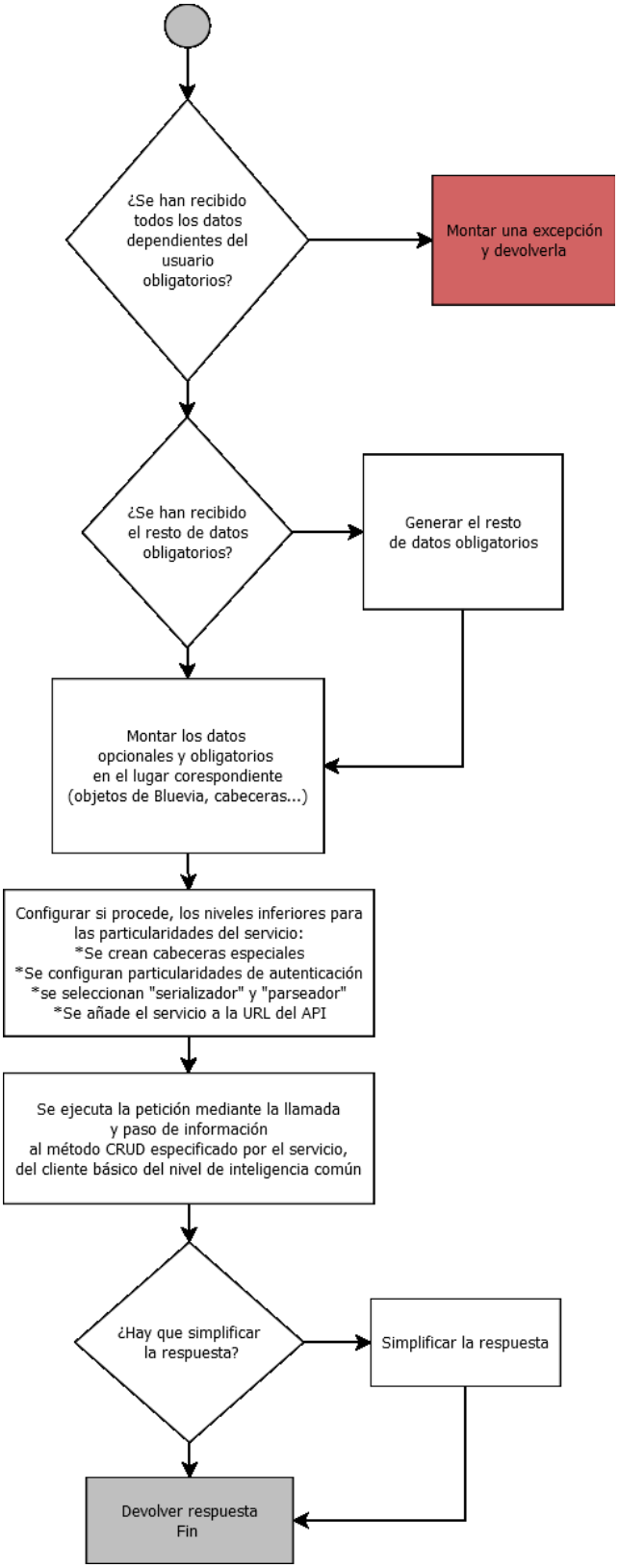


Figura 3-18 Flujo de la invocación de un servicio desde la perspectiva del nivel de inteligencia de API

3.3.3.2 Cliente final

En la parte más alta de la estructura del SDK, se encuentran los **Cliente finales**. Estos clientes hacen las veces de interfaz de uso del SDK, recuperando de la aplicación usuaria, el modo operativo en que desea funcionar -*Test, Live, Sandbox*-, sus datos de autenticación -*consumerKey* y *consumerSecret*-, y los datos que la autorizan a realizar peticiones en nombre del usuario final -*Tokens* o certificado SSL-.

Además, por cada servicio u opción especial de servicio existentes -desarrollados en los módulos base y extra del cliente común de API- , se expondrá un punto de entrada para invocarlos, requiriendo los datos dependientes del usuario para ello.

Como parte de las decisiones tomadas en grupo para aunar las interfaces, a diferencia de lo que ocurría en muchos servicios de 1.5, se decide que finalmente no se pasarán objetos complejos de *Bluevia* como parámetros para invocar los servicios, sino que se recibirán datos simples, y será la librería la encargada de generar los objetos para las peticiones.

De modo que para facilitar al usuario el acceso a los servicios de *Bluevia,* se ofrecerán más puntos de entrada o servicios virtuales, que servicios reales ofrece la plataforma.

A continuación se muestran tablas de la correspondencia entre los puntos de entrada y los servicios de acceso libre, en las figuras: Figura 3-19, Figura 3-20, Figura 3-21, Figura 3-22, Figura 3-23, Figura 3-24 y Figura 3-25.

Nota: en estas tablas no se consideran los servicios a los que se accede por métodos sobrecargados, sino a los que se accede por distintos métodos.

Servicio virtual ofrecido por el cliente de *OAuth*	**Servicio real Invocado**	**Particularidad**
`GetRequestToken`	`Oauth/getRequestToken`	Funcionamiento normal, sin cabeceras particulares de *Bluevia*, y con callback: URL u oob.
`GetRequestTokenSMSHandshake`	`Oauth/getRequestToken`	En vez de una URL de callback, se usa un MSISDN para recibir el código de verificación en un dispositivo móvil.
`GetAccessToken`	`Oauth/getAccessToken`	Funcionamiento normal.

Figura 3-19 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *OAuth*, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de *Advertising*	**Servicio real Invocado**	**Particularidad**
`GetAdvertising2L`	`Advertising/simple/requests`	El servicio se invoca sin autorización de usuario, tan solo con los *API Keys*. El campo *country* es obligatorio.
`GetAdvertising3L`	`Advertising/simple/requests`	El servicio se invoca con autorización de usuario, *API Keys + Tokens.*

Figura 3-20 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Advertising*, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de *Directory*	Servicio real Invocado	Particularidad
GetUserInfo	Directory/*/UserInfo	Funcionamiento normal.
GetAccessInfo	Directory/*/UserInfo/UserAccessInfo	Particularización del primero.
GetPersonalInfo	Directory/*/UserInfo/UserPersonalInfo	Particularización del primero.
GetProfileInfo	Directory/*/UserInfo/UserProfile	Particularización del primero.
GetTerminalInfo	Directory/*/UserInfo/UserTerminalInfo	Particularización del primero.

Figura 3-21 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Directory*, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de *Location*	Servicio real Invocado	Particularidad
GetLocation	Location/TerminalLocation	Funcionamiento normal.

Figura 3-22 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Location*, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de SMS	Servicio real Invocado	Particularidad
Send	Outbound/SMS/requests	Funcionamiento normal. Puede dispararse una notificación.
GetDeliveryStatus	Outbound/SMS/requests/*/deliverystatus	Funcionamiento normal.
GetAllMessages	Inbound/SMS/*/messages	Funcionamiento normal.
StartNotification	Inbound/SMS/subscriptions	Funcionamiento normal.
StopNotification	Inbound/SMS/subscriptions/*	Operación Delete, sobre una particularización del anterior.

Figura 3-23 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de SMS, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de MMS	Servicio real Invocado	Particularidad
Send	Outbound/MMS/requests	Funcionamiento normal. Puede dispararse una notificación.
GetDeliveryStatus	Outbound/MMS/requests/*/deliverystatus	Funcionamiento normal.
GetAllMessages	Inbound/SMS/*/messages	Funcionamiento

		normal.
GetMessage	Inbound/SMS/*/messages/*	Funcionamiento normal.
GetAttachment	Inbound/SMS/*/messages/*/attachments/*	Funcionamiento normal.
StartNotification	Inbound/SMS/subscriptions	Funcionamiento normal.
StopNotification	Inbound/SMS/subscriptions/*	Operación Delete, sobre una particularización del anterior.

Figura 3-24 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de MMS, y los servicios invocados de la plataforma *Bluevia*.

Servicio virtual ofrecido por el cliente de *Payment*	**Servicio real Invocado**	**Particularidad**
GetPaymentRequestToken	Oauth/getRequestToken	Con cabeceras particulares de *Bluevia*, y con callback: URL u oob.
GetPaymentAccessToken	Oauth/getRequestToken	Funcionamiento normal.
SetTokens	ninguno	Prepara al cliente para realizar una nueva transacción económica.
Payment	Payment/payment	Funcionamiento normal. Puede dispararse una notificación.
GetPaymentStatus	Payment/getPaymentStatus	Funcionamiento normal.
CancelAuthorization	Payment/cancelAuthorization	Funcionamiento normal.

Figura 3-25 Tabla de correspondencia entre servicios virtuales ofrecidos por el cliente de *Payment*, y los servicios invocados de la plataforma *Bluevia*.

3.4 DISEÑO DE BAJO NIVEL

3.4.1 Herencia en los módulos diseñados

La estructura diseñada en el capítulo anterior, se implementará en lo posible usando los mecanismos de herencia de clases -que en ***C#*** es herencia simple-.

De modo que se establece una cadena de herencia, donde los bloques situados más alto en la estructura -más cerca de la interfaz con el usuario-, extienden a las clases situadas por debajo.

Se protegen los métodos que vayan a ser heredados por otras clases, y se privatiza el resto.

Solo se da acceso público a los métodos que vayan ser usados por elementos fuera de la cadena de herencia.

3.4.1.1 Herencia en los módulos de inteligencia de *Bluevia*

Los clientes diseñados a partir del nivel de inteligencia común se encadenan en la forma mostrada en la Figura 3-26.

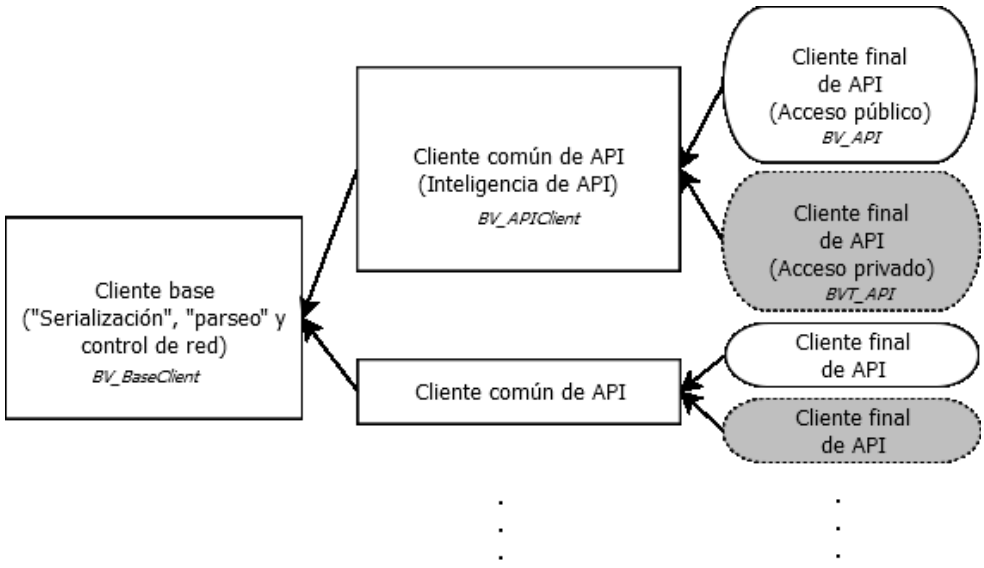


Figura 3-26 Diagrama de herencia entre clientes de *Bluevia*

Se dan tres salvedades a esta regla:

- ***Messagery***: es probablemente el API más complejo de los existentes en *Bluevia*; ya que engloba a dos API distintos en objetivo -SMS y MMS-, aunque muy parecidos en forma. En los que los servicios están divididos en dos grupos:
    - Servicios con el objetivo de enviar mensajes a un dispositivo móvil (*Mobile Terminated* o MT por sus siglas en Inglés-. Que necesitan de autorización de usuario, y por tanto son todos *3legged.*

- Servicios con el objetivo de recuperar de un buzón, mensajes enviados por dispositivos móviles -*Mobile Originated* o MO por sus siglas en Inglés-. Que no necesitan la autorización del usuario, y por tanto son todos *2legged*.

Así, que se establece una separación MT/MO, tanto conceptual como funcionalmente, de más alto nivel que la propia separación entre SMS/MMS. Por lo que se varía el esquema de herencia general para este API de la forma mostrada en la Figura 3-27.

Como puede observarse, el sistema de herencia no varía más que la pequeña concatenación de un eslabón adicional, que aporta la separación conceptual para subdividir SMS y MMS, en MT y MO.

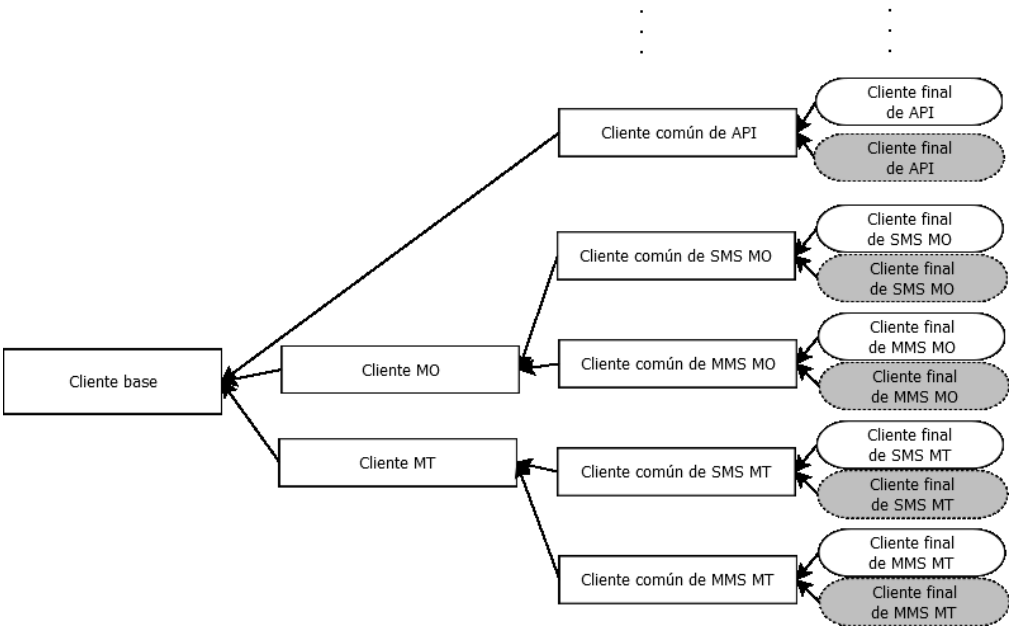


Figura 3-27 Diagrama de herencia para los API de Mensajería

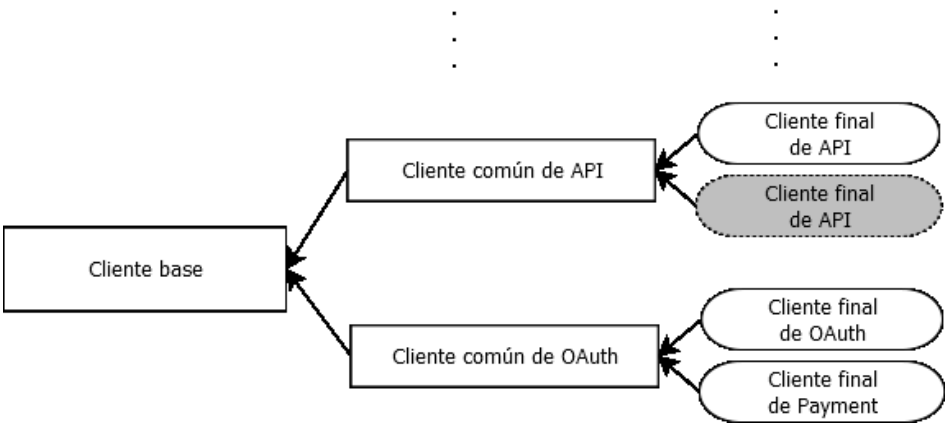


Figura 3-28 Diagrama de herencia para los API de Payment y OAuth.

- ***Payment***: Este servicio necesita de seguridad adicional; por lo que está provisto de un mecanismo de autorización más exigente en su versión pública, que requiere un uso intensivo y personalizado del API de *OAuth*. Lo que hace que no exista funcionalidad común con su versión privada.

  Por ello, para este API se elimina el cliente común. Y se hace extender al cliente final, dado que usa operaciones especiales de *OAuth*, del propio cliente común de *OAuth*, como se ha mostrado en la figura Figura 3-28.

- ***OAuth***: Este API se usa para autorizar aplicaciones y transacciones solo de acceso público. Por lo que no tiene cliente final de acceso privado.

3.4.1.2 Herencia para los módulos de conexión a la red

Para los bloques diseñados en el nivel de acceso a la red; dado que se quiere mantener una cierta modularidad para el caso de tener que cambiar de tecnologías; el esquema de herencia es más complicado.

En primer lugar, se define una interfaz que describe un conector de red genérico que solo ofrece las operaciones CRUD: será implementada como ***IBV_Connector***. Y dado que su funcionamiento es genérico no entra en el campo de la autenticación de peticiones.

Por otro lado se define una interfaz que describe una operación genérica para autenticar peticiones -para que sea implementada por tecnologías que realicen tal función-, ***IBV_Auth***.

Como el módulo de HTTPS (***HTTPConnector***) debe de usar operaciones CRUD, y autenticación; implementará ambas interfaces. Aunque marcará el método de autenticado como abstracto para que lo implemente la clase hija (ya que el sistema de autenticación también puede variar en un futuro).

Por último, el conector que engloba toda la funcionalidad del nivel de red (***BV_Connector***), hereda toda la funcionalidad de la clase ***HTTPConnector*** e implementa finalmente la autenticación; que para el caso de la versión 1.6 de *Bluevia* se realiza mediante *OAuth*. Como la autenticación mediante *OAuth* necesita operaciones particulares, este conector, implementa también una interfaz que las describe: ***IBV_OAuth***. Todo esto se ilustra en la Figura 3-29.

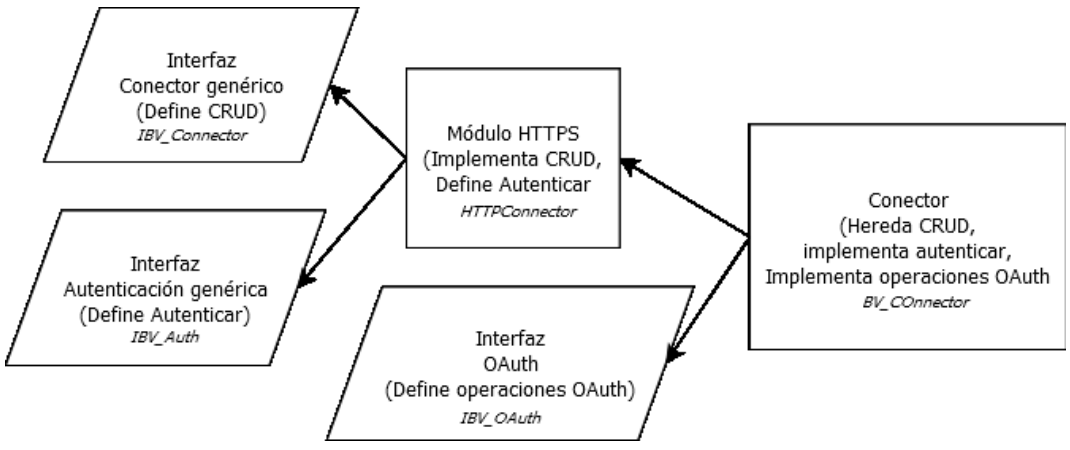


Figura 3-29 Diagrama de herencia entre conectores de *Bluevia*

3.4.2 Agrupando la funcionalidad

3.4.2.1 Integración de la funcionalidad al completo

Como acaba de verse en los puntos anteriores, la funcionalidad de acceso a la red bajo las especificaciones de *Bluevia*, queda englobada en el elemento ***BV_Connector.*** Mientras que la funcionalidad relativa a la inteligencia de los servicios, queda englobada en los **Clientes finales** (***BV_API*** y ***BVT_API***).

Para integrar ambas funcionalidades, pero -siendo consecuente con la directriz de diseño de "modularizar" el proyecto- dado que están conceptualmente separadas, en vez de recurrir a la herencia, se compondrán las clases. De modo que cada cliente contenga un conector -un cliente con la inteligencia de *Bluevia* contenga a un conector con la tecnología de acceso a la red de *Bluevia*-. Esto se ilustra en la Figura 3-30.

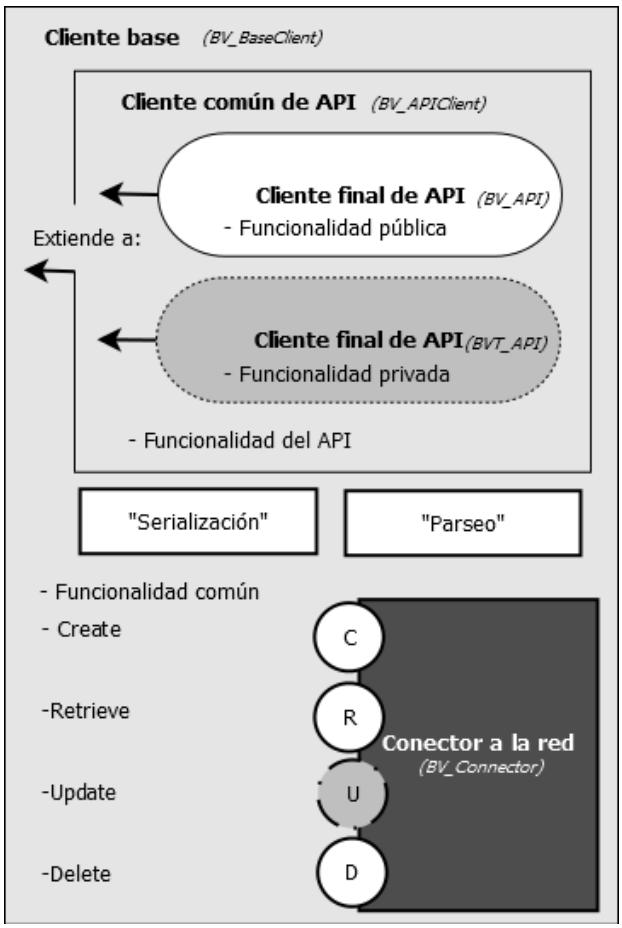


Figura 3-30 Esquema de integración de la funcionalidad del proyecto.

3.4.2.2 Inicialización del modo de acceso

La decisión entre los dos modos de acceso posibles a la hora de acceder a los servicios, se realizará a la hora de instanciar el cliente del API deseado, mediante el paso de parámetros -modo operativo, APIKeys y *Tokens o* certificado SSL- al constructor.

Dado que ***C#*** obliga a los constructores a ser públicos, y nuestra intención para simplificarle el uso al usuario es que solo pueda instanciar funcionalmente **Clientes finales**; el constructor con los parámetros descritos solo existirá en los **Clientes finales**, mientras que el resto de clases padre, contendrán un método protegido de inicialización.

Será en estos métodos de inicialización, donde se vayan generando las URL que apunten al *endpoint* adecuado, donde se instancien los "**Serializadores**" y "**Parseadores**" que necesite el API, y en general, donde se prepare al cliente con toda la información común concerniente al API.

En el nivel de acceso a red, el constructor con los datos de autorización y autenticación -la operativa de acceso se gestiona en los niveles de inteligencia- solo existirá en la clase más superficial, ***BV_Connector***.

Todo esto se ilustra en la figura Figura 3-31.

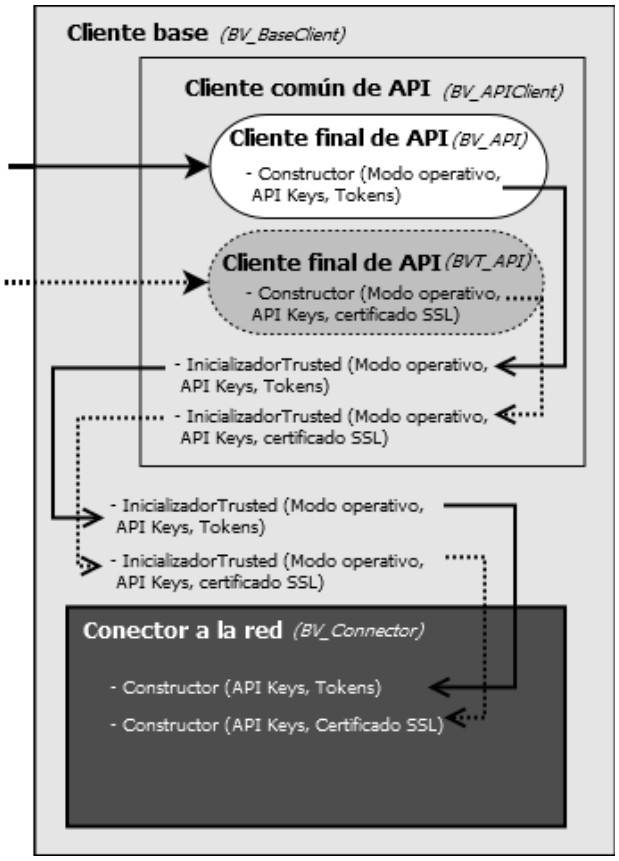


Figura 3-31 Esquema de inicialización encadenada.

3.5 RESUMEN DEL DISEÑO

Atendiendo a las directrices extraídas a partir del análisis de requisitos -Figura 3-6-, el grupo de desarrollo realiza un diseño básico del SDK basado en un conjunto de clientes especializados que poseen la inteligencia para acceder al API de *Bluevia* al que representan; usando un conjunto cerrado de otros objetos -que les proporciona su herencia a partir Cliente Común- que gestionan funcionalidades más básicas: un conector, un "serializador", y un "parseador".

- El **Conector**: Será el elemento que tienda la comunicación con la plataforma. Ofreciendo al Cliente un interfaz de uso CRUD, y estructurado de tal forma que pueda adaptarse a distintos tipos de tecnologías de conexión con poco cambios de código.
- El "**Serializador**": Convertirá los objetos de entrada para la invocación de los servicios, en los cuerpos apropiadamente formateados los mensajes que vayan a enviarse.
- El "**Parseador**": Procesara el cuerpo de las respuestas obtenidas, para extraer los campos de información relevantes

Esta estructura será coherente con la del resto de SDKs del resto de leguajes para la versión 1.6, permitiendo además una sencilla escalabilidad.

Además el diseño permite eliminar las dependencias en librerías externas, lo que ofrecerá mayor flexibilidad en el mantenimiento y escalado.

4 IMPLEMENTACIÓN Y VALIDACIÓN DEL PROYECTO

En este bloque se describen la implementación adaptada al leguaje ***C#*** y validación, de la solución diseñada en el bloque anterior; así como la elaboración de la documentación del SDK, y el entorno *software*.

- En el punto 4.1, se describe brevemente todo el software utilizado durante el desarrollo del SDK.
- En el punto 4.2, se describen los elementos implementados para completar en el lenguaje ***C#***, lo diseñado en el bloque 3: DISEÑO DE LA VERSION 1.6 DEL SDK DE ACCESO A BLUEVIA PARA *.NET*.
- En el punto 4.3, se describe la solución implementada para funcionar como paquete de ejemplos de uso, de las librerías creadas.
- En el punto 4.4, se describe el proceso de validación de las librerías creadas, junto con algunas muestras de dicho proceso.
- En el punto 4.5, se describe los aspectos relacionados con la documentación del SDK; las consideraciones a la hora de documentar el código, así como el diseño y creación de las guías de uso.

4.1 SOFTWARE UTILIZADO PARA LA REALIZACIÓN DEL PAQUETE ENTREGABLE

Para la realización del proyecto, han sido necesarias las siguientes herramientas *software*:

4.1.1 Sistema operativo

1. ***Microsoft Windows 7***[16]***:*** A pesar de que las primeras versiones del SDK se desarrollaron en ***Windows XP*** de 32 bits como base, la versión que nos ocupa ha sido desarrollada al completo sobre el sistema operativo ***Windows 7*** **Professional** de 64 bits -en dos máquinas: una actualizada al día, y otra en su versión pre-lanzamiento-. En todos los casos, el sistema operativo tiene instalado el ***.NET Framework 4*** de ***Microsoft***.

   Dado que dicho *framework* -requisito para el desarrollo del SDK- corre sin problemas sobre cualquiera de los sistemas operativos citados, la elección por uno solo fue cuestión de comodidad.

4.1.2 Software para el desarrollo de la librería

2. ***Microsoft Visual Studio 2010***[17]***:*** En sus versiones ***Professional*** y ***Ultimate***. Este entorno de desarrollo integrado -IDE- para sistemas operativos ***Windows***, ofrece funcionalidades para el desarrollo de proyectos relacionadas con el entorno ***.NET***, sus diversos lenguajes de programación y servidores. Para este proyecto se han utilizado las relativas a la creación de proyectos en lenguaje ***C#***.

   La elección de la versión 2010, corresponde a la necesidad de usar el ***.NET framework 4***, soportado solo a partir de esta versión del IDE.

3. ***Fiddler 2***[18]***:*** *Proxy* de depuración de red para ***Windows***, gratuito, que permite la inspección en detalle del tráfico web.

   Ha sido usado de forma intensiva en el proyecto para verificar el correcto funcionamiento de la librería en la generación de: conexiones, cabeceras, y cuerpos de mensaje.

4. ***Whireshark***[19]***:*** *Proxy* de depuración de red multiplataforma, con licencia GPL[54], que permite la inspección en detalle de múltiples tipos de tráfico de red.

   Ha sido usado para la verificación de las conexiones de la librería *Trusted,* que usan SSL de dos vías.

4.1.3 Software para el desarrollo de la documentación

5. ***Doxygen***[20]***:*** Es un sistema multiplataforma, con licencia GPL, para generar documentación de proyectos *software* en varios formatos. Soporta varios lenguajes de programación, y ofrece muchas opciones para detallar la documentación en varios niveles.

   Con esta herramienta, se generan los manuales y guías HTML que se adjuntan en el paquete de entrega.

6. ***Notepad++***[21]***:*** Es un editor de código para ***Windows***, con licencia GPL, de manejo sencillo y soporte para múltiples lenguajes de programación.

   Se ha usado para escribir los documentos con los que ***Doxygen*** genera los manuales de uso del SDK, y editar los textos relativos al paquete, tales como el archivo de exclusión de ***GIT***, el MD5 etc…

7. ***Adobe Photoshop CS5***[22]***:*** Es un editor de imagen digital extraordinariamente potente, que permite desde procesar fotografías, hasta crear ilustraciones profesionales.

   Tan solo se ha usado en el proyecto, para editar las capturas de pantalla que ilustran los manuales de la documentación. La versión del programa ha sido consecuencia de la disponibilidad en la empresa.

4.1.4 Software para el control de versiones y entregas

8. ***GIT for Windows***[25]***: Git***[23] es un sistema de control de versiones, con licencia GPL, que facilita el desarrollo distribuido de proyectos. La herramienta ***GIT for Windows*** permite utilizar este sistema en los sistemas operativos de ***Microsoft***. Proporcionando una interfaz gráfica, y otra de consola.

   Git se ha usado para controlar la evolución del desarrollo de los SDKs, y mantener copias de seguridad de estos.

9. ***Dropbox***[26]***:*** Es un sistema de almacenamiento online, que permite la sincronización de archivos y carpetas entre varios equipos; a través de un interfaz web, o clientes para múltiples plataformas.

   Dado que en varias ocasiones el desarrollo de la aplicación se ha realizado desde fuera de las instalaciones de ***Telefónica I+D***, esta herramienta ha permitido la coordinación de los archivos.

10. ***WinSCP***[28]***:*** Es un cliente de SCP[27] y otros protocolos de transmisión de archivos, con licencia GPL, para ***Windows;*** que permite la copia de archivos entre máquinas remotas.

    Se ha usado para las entregas finales de los paquetes.

11. ***Winrar***[29]***:*** Es un programa de compresión de archivos, para varias plataformas, que soporta varios formatos y permite múltiples opciones de compresión.

Se ha usado para generar el paquete de entrega, en formato .ZIP.

12. ***WinMd5Sum***[31]: Es una herramienta para crear y comprobar sumas MD5[30], y así verificar que no se ha alterado un archivo descargado de internet.

    Se ha usado para generar el código MD5 con el que verificar los paquetes de entrega.

4.1.5 Otros

13. ***Mozilla Firefox***[32]***:*** Es un Navegador web multiplataforma desarrollado por ***Mozilla***, con licencia GPL.

    Con él se ha accedido al entorno de coordinación de *Bluevia*, para descargar las especificaciones de los servicios, dar soporte a clientes, y realizar parte de las comunicaciones del proyecto. Además se ha usado para comprobar el formato de la documentación generada del SDK.

14. ***Adobe Reader***[33]***:*** Es un visualizador gratuito de archivos .PDF creado por ***Adobe***.

    Ha sido utilizado para visualizar parte de los requisitos de la librería, ya que las especificaciones de los servicios de *Bluevia* se distribuían en formato .PDF.

15. ***Skype***[34]: Es un cliente de chat multiplataforma ampliamente utilizado, en propiedad de ***Microsoft***. Permite comunicación por texto, audio y voz; además del intercambio de archivos; entre varios participantes de una conversación.

    Usado de forma intensiva para coordinarse de manera distribuida con el resto del proyecto. Tanto para recibir indicaciones por parte del jefe del grupo, como para comunicarse con el resto de grupos de *Bluevia*; poder recibir soporte en el desarrollo y ofrecer soporte en pruebas.

16. ***Eclipse Indigo***[35] ***+ Apache Tomcat***[36]***: Eclipse*** es otro IDE multiplataforma y con licencia EPL, del que existen varias versiones enfocadas al desarrollo de varios tipos de proyectos y lenguajes. Para el proyecto se ha usado la versión *Indigo* enfocada al desarrollo de proyectos ***Java EE***, ya que se pretendía aprovechar parte del código de otro proyecto en curso realizado en este entorno.

    Apache es un servidor web, con licencia Apache, de amplia implantación en el mercado. Su versión ***Tomcat*** permite montar los servicios web a partir de *Servlets* ***Java***.

    Durante los primeros estadios del proyecto, se montó el servidor sobre el IDE, y se desarrolló un servicio web que emulaba parte del comportamiento del servicio *Advertising* de *Bluevia*; para probar el comportamiento de las librerías de ***.NET***, y las primeras funcionalidades del SDK.

4.2 IMPLEMENTACIÓN DE LAS LIBRERÍAS

El desarrollo de este SDK ha pretendido ajustarse fielmente al diseño alcanzado en común por el grupo de SDKs -descrito en el bloque anterior 3-, a pesar de que durante el proceso, el diseño original ha ido modificándose para adecuarse a los problemas que surgían en, y entre, los diversos lenguajes en los que se desarrollaban los SDK: problemas de tipos en ***PHP*** y ***Ruby***; librerías auxiliares inflexibles en ***Ruby***, ***Android*** y ***Java***; incluso problemas con la protección y herencia en ***Java*** y ***C#***.

4.2.1 Sistema de paquetes

La implementación de este SDK se comenzó realizando la distribución del sistema de paquetes -en general, dado que esa algo inherente a las convenciones de cada lenguaje en particular, no hay una distribución de paquetes en común-.

La funcionalidad, se agrupará realizando un empaquetamiento coherente con la herencia diseñada; aunque tratando de separar la común a varios API, de la específica de cada uno. De modo que para añadir en un futuro el acceso a los servicios de un nuevo API, solo tenga que añadirse un paquete nuevo relativo a dicha inteligencia.

Así, todo el código que solo afecte a un API, colgará del paquete referente a ese API, y el código que sea usado, o sea susceptible de ser usado en un futuro, por varios API, colgará de un paquete principal.

Quedando la estructura primaria de la siguiente forma:

- Un Paquete principal ***Core.***
- Y un paquete relativo a cada API existente:
    - Paquete de API ***OAuth***
    - Paquete de API ***Advertising***
    - Paquete de API ***Directory***
    - Paquete de API ***Location***
    - Paquete de API ***Messagery***
    - Paquete de API ***Payment***

En el primer nivel, los paquetes contendrán solo una clase ***Constants.cs*** que será donde se definan las constantes y tipos enumerados relativos al paquete -toda aquella información que solo vaya a ser leída-.

Este nivel se subdividirá en otros paquetes según la funcionalidad de las clases que contengan:

- Subpaquete de clientes ***Client:***

Donde se almacenan las clases con lógica de cliente relativa a los niveles **Común** y **de API**, y las interfaces que las definen. Los comunes en el paquete ***Core***, y los específicos de cada API en su paquete respectivo.

- Subpaquete de objetos ***Schemas:***

  En el que se almacenan las clases contenedoras de datos; tanto las que se codifican y envían en los mensajes cliente <-> servidor, como los que se usan internamente en el SDK, o se ofrecen en la interfaz de usuario (Salvo en algún caso de MMS, estas clases solo almacenan datos, y no tienen algoritmia).

  En los paquetes de las APIs, este subpaquete también contiene la definición XML -XSDs- de los objetos que se intercambiarán durante la comunicación cliente <-> servidor.

- Subpaquete de clases auxiliares y herramientas ***Tools:***

  De aquí colgarán las clases no meramente contenedoras, sin funcionalidad global; como las que "serializan" o "parasen" los mensajes o las que ejecutan la algoritmia de seguridad entre otras.

- Solo en el caso del paquete ***Core***, se da la subdivisión en el paquete ***Connectors:***

  Donde se almacenan las clases con lógica de cliente relativa al nivel de **Acceso a la red**, y las interfaces que las definen.

4.2.2 Descripción de clases

Se describen a continuación las clases del proyecto y sus elementos, de forma general. Para que el lector pueda hacerse una idea clara de la estructura y el funcionamiento de las librerías.

Las descripciones se agruparán según la separación de niveles descrita en el capítulo de diseño, de modo que pueda establecerse una relación directa con los argumentos expresados en este:

- **Clases del nivel de acceso a la red**. Contenidas en el paquete ***Core*** del proyecto, proporcionarán la funcionalidad de conexión, autenticación, envío y recuperación de mensajes, con el ***BV_Connector*** como clase principal.

- **Clases del nivel de inteligencia común**. Contenidas también en ***Core***, desarrollan la funcionalidad común requerida por los servicios para acceder a *Bluevia*, con todas las opciones centralizadas en la clase ***BV_Client***.

- **Clases de nivel de inteligencia de API**. Con las clases referentes a cada servicio, contenidas en su respectivo paquete: ***Advertising, Directory, Location, Messagery, Payment*** y ***OAuth***. Establecen los requisitos particulares de cada API, usando la funcionalidad de los otros dos niveles, y ofrecen una interfaz de uso de la librería de cara al exterior.

Aunque los clientes finales, que hacen de interfaz de cara al exterior son las clases ***BV_API, y BVT_API*** -para la versión privada-, las clases principales de estos paquetes serán las ***BV_APIClient***.

4.2.2.1 Clases del nivel de acceso a la red

Listadas de las más generales a las de funcionalidad más limitada, y contenidas en el paquete ***Core*** del proyecto, proporcionarán la funcionalidad de conexión, autenticación, envío y recuperación de mensajes, con el ***BV_Connector*** como clase principal. Se muestra un esquema de estas, sus relaciones y métodos en la Figura 4-1.

a. Conectores e interfaces

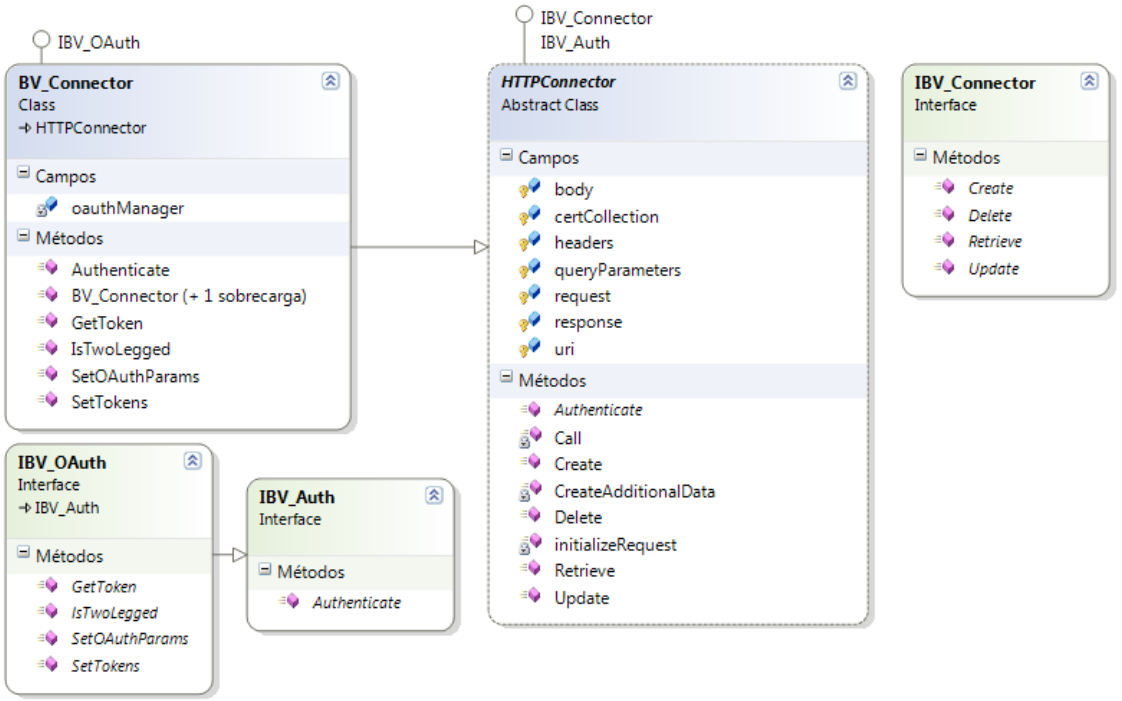


Figura 4-1 Diagrama de conectores e interfaces del nivel de acceso a la red.

- **BV_Connector** *(Bluevia.Core.Connectors.BV_Connector)*

  Esta clase engloba toda la funcionalidad del nivel de acceso a la red. Tiene código propio para autenticar las peticiones de *Bluevia,* ya que implementa la interfaz ***IBV_OAuth***; y hereda del ***HTTPConnector*** la funcionalidad para montar dichas peticiones, enviarlas y recuperar las respuestas.

  Como puede apreciarse en la Figura 4-1, esta clase ofrece dos constructores sobrecargados; uno para instanciar un conector de acceso a los servicios privados –*Trusted-*, y otro para acceder a los servicios públicos –*Non-Trusted-*. Además de los identificadores de aplicación –*Consumer Key,* y *Secret-*, el primero de los constructores también demandará como parámetro una colección de certificados que servirá para verificar la legitimidad de las peticiones por medio de una conexión SSL de dos vías.

Mientras que el segundo constructor requerirá los *Tokens* que demuestren que el usuario ha autorizado a la aplicación a realizar peticiones en su nombre. Estos *Tokens* podrán ser omitidos en caso de que la aplicación que vaya a acceder a *Bluevia,* sea de funcionamiento *2legged.*

Nota: Cada cliente de la librería que cree un usuario, usará una instancia diferente de BV_Connector.

- El ***BV_Connector*** contiene un objeto ***OauthManager*** que es el elemento donde cada cliente guarda los *Tokens* de autorización de la aplicación, y el encargado de generar la cabecera de autenticación de las peticiones.

Para controlar este elemento ***OauthManager***, el ***BV_Connector*** ofrece varios métodos. Siendo los más destacados:

- *SetTokens,* permite añadir, eliminar o cambiar, los *Tokens* de autorización de la aplicación. De modo que un mismo cliente de acceso a los servicios públicos, pueda realizar peticiones *2 o 3legged* con diferentes autorizaciones (Pej: esto es necesario en el caso del servicio de Oauth, cuando durante el proceso de autorización de una aplicación, el cliente deberá realizar peticiones sin *Tokens*, y con *Tokens* temporales, en operaciones sucesivas).
- *GetToken*, permite recuperar el *Token* público en uso en ese momento, para poder generar identificadores y rutas para los servicios.
- *SetOAuthParams,* permite suministrar al ***OauthManager*** parámetros extra, necesarios por algunos servicios especiales, para que los use en la generación de la cabecera de autenticación.
- *Authenticate*, es un método invocado en todas cada una de las peticiones a *Bluevia*, que configura el ***OauthManager*** según la petición a realizar, y ejecuta su método principal para que genere la cabecera de autenticación adecuada; una vez generada, la añadirá a la petición en ciernes.
- *IsTwoLegged,* método declarado con la idea de escalar más adelante las librerías. Invocado por el API de *Advertising*, para conocer en qué modo debe realizar las peticiones.

- **HTTPConnector** *(Bluevia.Core.Connectors.HTTPConnector)*

Esta clase abstracta implementa la funcionalidad de un conector de red para *Bluevia* descrita por la interfaz ***IBV_Connector***, particularizada para la tecnología HTTP. Proporcionando a la clase ***BV_Connector*** las capacidades de conexión, envío y recuperación de mensajes, necesarias para invocar los servicios.

Más allá de los nombres de algunos objetos usados, esta clase es independiente de la lógica de *Bluevia*, ya que no gestiona los identificadores ni las particularidades de los servicios -esa funcionalidad esta delegada en la clase hija ***BV_Connector***-, de modo que podría ser usada directamente en otras aplicaciones, con modificaciones mínimas.

La clase ***HTTPConnector*** contiene los siguientes objetos para administrar las peticiones:

- *HttpWebRequest* ***request***. Es el elemento proporcionado por el Framework de ***.NET*** para ***C#***, que permite el establecimiento y uso de una conexión HTTP/HTTPS. Se instancia y apunta -de forma estática- en el método *initializeRequest*; y en el método *Call,* se montan los campos de los mensajes, se envían las peticiones y recuperan las respuestas, siempre sobre este objeto.

- *HttpWebResponse* ***response***. Es el elemento que contendrá la respuesta o el error HTTP. Se recupera desde el objeto ***request***, siempre que se haya podido realizar la petición. Si la respuesta contiene un cuerpo, este se procesa en el método *CreateAdditionalData*.

- *Uri* ***uri***. Objeto que contiene la dirección del servicio a invocar, junto con los parámetros de URL necesarios. Se genera en el método *initializeRequest* a partir de los parámetros recibidos en los métodos CRUD: *uri* y *parameters*.

- *byte[]* ***body***. *Array* de *Bytes* que contiene el cuerpo del mensaje HTTP que se pretende enviar. Se recibe como parámetro de los métodos CRUD: *Create* y *Update,* y se monta en el método *Call*, sobre un *stream* que dispone el objeto ***request***.

  Nota: Inicialmente el objeto *body* era un *array* de caracteres, para una depuración más sencilla. Pero esto hacía imposible una codificación correcta de los archivos multimedia enviados con MMS, por lo que finalmente se cambió a un *array* de *Bytes*.

- *Dictionary<string, string>* ***headers***. Colección de pares: "Nombre de cabecera HTTP-Valor", que definen al mensaje que será enviado al servicio. Se recibe como parámetro de los métodos CRUD: *Create* y *Update*, y no contendrá las cabeceras *Authorization* ni *Content-Length*.

  Se monta en el método *Call*, sobre el campo apropiado del objeto ***request*.**

- *X509CertificateCollection* ***certCollection***. Es la colección de certificados de sistema necesarios para poder establecer una conexión SSL de dos vías. Si existe, querrá decir que el servicio a invocar es de carácter privado, y se usará para establecer la conexión en el método *Call*.

- *Dictionary<string, string>* ***queryParameters.*** Elemento donde se almacenan los parámetros recibidos para realizar una petición.

La clase ***HTTPConnector***, ofrece como interfaz de uso los cuatro métodos CRUD:

- *Create*, *Retrieve*, *Delete* y *Update*. Todos ellos reciben como parámetros un *string: uri*, con la dirección del *endpoint* del servicio a invocar; y una colección de pares "Parámetro de URL-Valor": *parameters*, que contiene los parámetros de URL necesarios para invocar el servicio. Ambos generados en el nivel superior de la librería por los clientes de inteligencia de API.

  Los métodos *Create* y *Update*, contendrán además dos parámetros extra: el *array* de *Bytes body,* que contendrá el cuerpo del mensaje generado por los "serializadores" del nivel de inteligencia común; y una colección de pares "Nombre de cabecera HTTP-Valor": *headers*, generados en el nivel superior de la librería por los clientes de inteligencia de API, que aportan la "metainformación" del mensaje.

En primer lugar comprueban que los parámetros recibidos son los esperados para la operación, tras ello, inicializan el objeto ***request***, establecen en el método HTTP correspondiente, autentican la petición, y finalmente invocan al método *Call* para devolver el resultado.

Además de estos métodos públicos, la clase contiene varios privados más para realizar el cometido de los anteriores:

- *initializeRequest*, este método es llamado por los métodos CRUD para instanciar el objeto ***request*** apuntándolo al *endpoint* especificado, y copia los parámetros con la información de la petición, en los campos del objeto ***HTTPConnector***.
- *Call,* este método es invocado también por los métodos CRUD una vez se ha apuntado y autenticado el objeto ***request***. Se encarga de montar los datos de la petición, realizar la conexión, y recuperar la respuesta. Finalmente en caso de respuesta exitosa o error HTTP, invoca el método *CreateAdditionalParameters* para procesar la respuesta y completar el objeto ***GenericResponse*** a devolver.
- *CreateAdditionalParameters,* este método es invocado por el método *Call* para procesar la respuesta o error HTTP y extraer tanto el cuerpo como las cabeceras, para montar un objeto ***AdditionalResponseData***.

- **IBV_Oauth** *(Bluevia.Core.Connectors.IBV_Oauth)*

Esta interfaz, implementada por la clase ***BV_Connector***, define la funcionalidad que debe implementar un conector de red para *Bluevia* que autentique usando *Oauth*. Dado que es una interfaz de autenticación utiliza también la interfaz ***IBV_Auth***.

Define los métodos:

- *SetOauthParams*. Para proporcionar las opciones especiales de *Bluevia* desde el cliente.
- *SetTokens*. Para cambiar el comportamiento del conector, pasándolo de *2legged* a *3legged*, o cambiar la autorización de usuario.
- *GetToken*. Para poder recuperar el *Token* que hace las veces de identificador de usuario final, para las aplicaciones de acceso libre.
- *IsTwoLegged*. Que informa del comportamiento actual del conector. Si está actuando como intermediario de una aplicación sin necesidad de autorizaciones, o por el contrario lo hace de una aplicación autorizada.

- **IBV_Auth** *(Bluevia.Core.Connectors.IBV_Auth)*

Esta interfaz aunque es declarada en la clase ***HTTPConnector*** es finalmente implementada en la clase ***BV_Connector***. Define la funcionalidad que tiene que cumplir un conector de red que necesite autenticar sus peticiones, mediante su método:

- *Authenticate*. Que describe la necesidad de autenticar las peticiones.

- **IBV_Connector** *(Bluevia.Core.Connectors.IBV_Connector)*

  Esta interfaz es implementada por la clase ***HTTPConnector*** describiendo la funcionalidad que debe cumplir un conector para acceder a la red de *Bluevia*, a través de los métodos CRUD:

  - *Create*. Método que define la creación de un recurso. Será usado como equivalente a la operación POST en HTTP, para enviar documentación al servicio invocado.
  - *Retrieve*. Método que define la recuperación de un recurso. Será usado como equivalente a la operación GET en HTTP, para pedir información al servicio invocado.
  - *Update*. Método que define la actualización de un recurso. No es implementado en la versión 1.6 de *Bluevia*.
  - *Delete*. Método que define la eliminación de un recurso. Será usado como equivalente a la operación DELETE en HTTP, para dar de baja el servicio invocado.

b. Herramientas y clases auxiliares del nivel de acceso a la red

Estas clases contienen la lógica necesaria para cumplir cometidos de ámbito específico dentro de las necesidades del acceso a la red. Se muestra un esquema de estas, sus relaciones y métodos en la Figura 4-2.

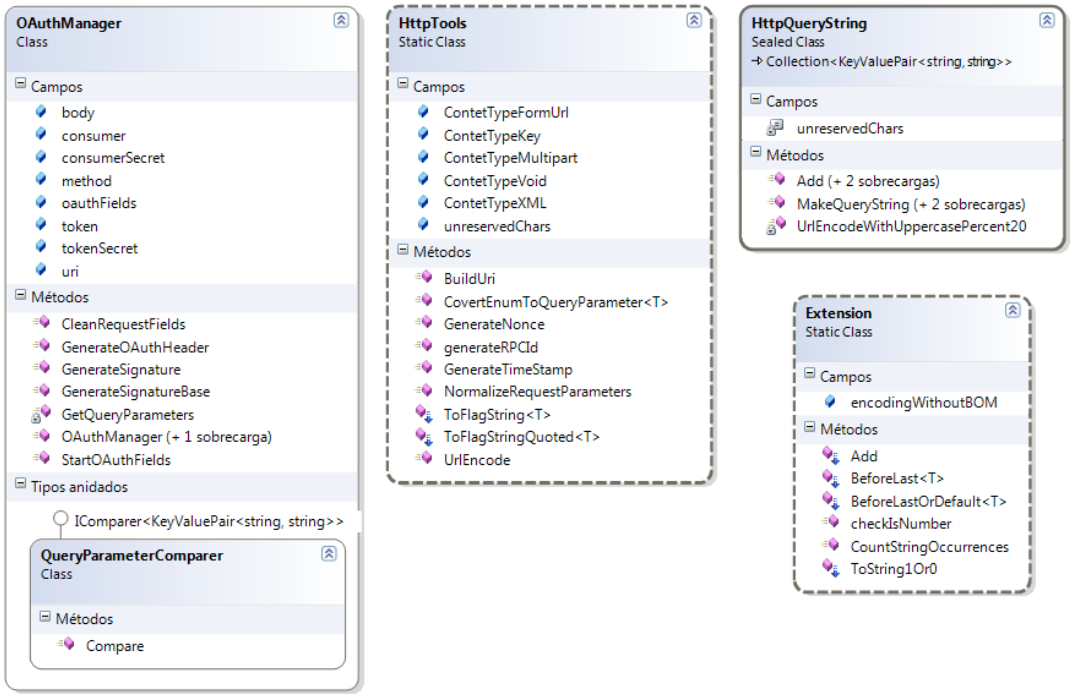


Figura 4-2 **Diagrama de clases auxiliares del nivel de acceso a la red**

- **OauthManager** *(Bluevia.Core.Tools.OauthManager)*

  Esta clase autentica peticiones a los servicios, siguiendo la especificación de la tecnología *Oauth* con algunas variaciones para la plataforma *Bluevia*, mediante la generación de la cabecera HTTP: *"Authorization: Oauth"*. Esta cabecera, se rellena con pares: campo_oauth="valor", de los que el más importante es la firma.

  Nota: Cada Conector usará tan solo una instancia de esta clase que irá modificando sus valores contenidos, durante todo su periodo de vida.

  Para rellenar esta cabecera, el objeto dispone de los siguientes campos:

  - `string` ***consumer*** y `string` ***consumerSecret***. Ambos atributos Identifican de forma única a la aplicación que quiere hacer uso del servicio. Son pasados como parámetro durante la instanciación del **Cliente de API**, y finalmente almacenados en el objeto ***OauthManager***. El primero viajará como valor junto al campo *oauth_consumer_key*; mientras que el segundo se usa como parte de la clave de codificación para realizar la firma de la cabecera *"Authorization: Oauth"*, junto al campo *oauth_signature*.

  - `string` ***token*** y `string` ***tokenSecret***. Ambos atributos son el producto de la autorización por parte del usuario, para que la aplicación pueda invocar servicios en su nombre. Son pasados como parámetro durante la instanciación del cliente de API y finalmente almacenados en el objeto ***OauthManager***, aunque pueden ser modificados a posteriori para cumplir con los requisitos de seguridad de algunos servicios. De existir –no existen en aplicaciones *2legged-*, al igual que en el caso de las claves *consumer*, el primero viajará como valor junto al campo *oauth_token*; mientras que el segundo se usa como parte de la clave de codificación para realizar la firma de la cabecera *"Authorization: Oauth"*, junto al campo *oauth_signature*.

  - `Dictionary<string, string>` ***oauthFields***. Colección para almacenar los campos que componen la cabecera *"Authorization: Oauth"*. Se inicializa en cada petición con los campos básicos, al principio de la autenticación de esta si el servicio no necesita de campos especiales. O, si la petición lo requiere, se le pasa una colección ya creada con los campos especiales desde el cliente del servicio –con el método del ***BV_Connector***: *SetOAuthParams-*, añadiéndole los básicos a posteriori.

  - `Uri` **`uri`** y `string` **`method`**. Ambos valores, representan el *endpoint* y el tipo de petición respectivamente, para el servicio a invocar. Se utilizan como parte de la base para generar la firma *–oauth_signature-* de la cabecera "*Authorization: Oauth".*

  - `string` ***body***. Solo en el caso de que la invocación al servicio deba de contener un cuerpo en formato *FormURLEncoded*, este se utilizará como parte de la base para generar la firma *–oauth_signature-* de la cabecera "*Authorization: Oauth".*

  El objeto ***OauthManager*** dispone también de los siguientes métodos para generar el contenido de la cabecera de autenticación:

- Dos *constructores* sobrecargados, que reciben por parámetro los datos que identifican a los actores demandantes del servicio: el primero con los *consumers* que identifican a la aplicación, y los *Tokens* que identifican al usuario. Y el segundo solo con los *consumers* para el caso de que la aplicación sea *2legged*.

- *StartOauthFields*. Inicializa o completa –en caso de que la petición requiera campos especiales en la cabecera *"Authorization: Oauth"*-, la colección de pares "Campo Oauth-Valor" del atributo *oauthFields*.

- *GenerateOauthHeader*. Método que invoca al método *GenerateSignature* para generar, y añadir a la lista de pares de *oauthFields*, el valor del campo: *oauth_signature*. Y crea, ordenando los pares, el contenido de la cabecera *"Authorization"*.

- *GenerateSignature*. Este método invoca al método *GenerateSignatureBase* para generar la base para la firma de la cabecera "*Authorization: Oauth"*. Tras ello, monta la clave de cifrado y genera la firma siguiendo lo especificado en el RFC de Oauth 1.0 [12].

- *GenerateSignatureBase*. Este método sigue el RFC de Oauth 1.0 [13], para añadir, codificar y ordenar adecuadamente los elementos de la petición necesarios para generar la base de la firma de *Oauth*.

  Para la ordenación, utiliza elemento interno al ***OauthManager: QueryParameterComparer***.

- *CleanRequestFields.* Elimina los campos susceptibles de provocar erro en futuras peticiones.

- **QueryParameterComparer** *(Bluevia.Core.Tools.OauthManager - > QueryParameterComparer)*

  Este elemento interno al ***OauthManager*** y recuperado de las versiones antiguas del SDK, implementa la funcionalidad de comparar objetos de la interfaz del *framework:* ***IComparer***, para adecuarlo al comportamiento de realizar la ordenación de parámetros de las peticiones, a la hora de generar la firma de la cabecera de *OAuth*.

- **HttpTools** *(Bluevia.Core.Tools.HttpTools)*

  Esta clase estática ofrece al resto de las clases de la librería, pequeños métodos relacionados con el protocolo HTTP: Generadores de valores para algunos campos, y métodos para formatear parámetros de URL, etc.

  - *BuildUri*. Este método recibe la dirección de un *endpoint* y una colección de pares "Parámetro de URL-Valor", para construir con la ayuda de la clase sellada HTTPQueryString, el objeto *Uri* necesario en la instanciación del elemento ***request*** del ***HTTPConnector***.

  - *GenerateNonce* y *generateRPCId*. Estos dos métodos generan un UID de formato adecuado al cometido de cada uno.

El primero es invocado en el método *StartOAuthFields* del objeto ***OauthManager***, para generar el valor del campo *"Nonce"* de la cabecera *"Authorization: Oauth".*

Y el segundo es invocado en los clientes de *Payment* –el único API en la versión 1.6 que usa RPC-, para generar el identificador de transacción RPC.

- *GenerateTimeStamp*. Este método se utiliza para calcular los segundos transcurridos desde el 1 de enero de 1970, y rellenar con dicho número el valor del campo *oauth_timestamp* de la cabecera *"Authorization: Oauth"* y crear las fechas de las transacciones del API de *Payment.*

- *ConvertEnumToQueryParameter<T>*. Este método recibe un *array* de enumerados del tipo "T" genérico y haciendo uso de la funcionalidad de ***C#*** "*Reflection",* lo convierte en un *string* de las opciones enumeradas seleccionadas, separadas por comas, para que funcione como valor de parámetro de URL.

  Es usado en los métodos del API de directorio, para seleccionar los campos o sets de información deseados.

- *NormalizeRequestParameters.* Método que recibe una lista ordenada de parejas "Parámetro-Valor" y la codifica adecuadamente usando el método *UrlEncode* para cumplir con la especificación de Oauth.

- *UrlEncode.* Realiza una codificación *"Percent-Encoding",* que a diferencia de la se ofrece en los métodos del *framework* de ***.NET***, escapa los caracteres especiales con mayúsculas, tal y como lo especifica el RFC de Oauth [14].

- *ToFlagString<T>.* Recuperado de las versiones anteriores de *Bluevia*, proporciona la misma funcionalidad que *ConvertEnumToQueryParameter*, siempre que los enumerados tengan las opciones marcadas en binario. Mantenido para temas de codificación y ordenación de cadenas en alguna funcionalidad futura.

- *ToFlagStringQuoted<T>.*Invoca a *ToFlagString*, entrecomillando los valores.

- **HttpQueryString** *(Bluevia.Core.Tools.HttpQueryString)*

  Clase sellada que extiende la funcionalidad de los objetos colección, con la finalidad de completar el objeto ***uri*** –necesario para instanciar el objeto ***request*** de la clase ***HTTPConnector***- con los parámetros de la petición.

  Sobrecargando varios métodos para tal fin:

  - *Add.* Varios métodos sobrecargados para recibir los parámetros de varias formas posibles, parejas, objetos o listas.

  - *MakeQueryString.* Varios métodos sobrecargados para recibir el objeto *uri* a completar, y los parámetros en varios formatos. Se encarga de codificar correctamente los parámetros, e integrarlos en el *uri.*

- *UrlEncodeWithUppercasePercent20.* Pequeña funcionalidad mantenida de las versiones anteriores del SDK con vistas a proporcionar funcionalidad futura en temas de codificación y ordenación de cadenas.

- **Bluevia.Core.Tools.Extension**

  Esta clase estática provee funcionalidad aislada al nivel de inteligencia común, y de API.

  - *Add*. La finalidad de este método es la reescribir la funcionalidad del método *Add* de las colecciones de ***C#***, para facilitar la inserción de pares, de modo que solo se añadan a la colección si tanto la clave como el valor tienen contenido.
  - *checkIsNumber*. Este método verifica que el contenido de un *string* sea numérico.
  - *CountStringOccurences*. Método que devuelve el número de apariciones de un patrón, en un texto.
  - *BeforeLast, BeforeLastOrDefault, encodingWithoutBoom, toString1Or0.* Pequeñas funcionalidades mantenidas de las versiones anteriores del SDK con vistas a proporcionar funcionalidad futura en temas de codificación y ordenación de cadenas.

c. Objetos contenedores del nivel de acceso a la red

El cometido de estas clases es el de contener la información necesaria para que el resto de clases del nivel puedan operar con ella. Pueden separarse en dos grupos:

Clases de constantes del nivel de acceso a la red

Se muestra un esquema de estas, sus relaciones y métodos en la Figura 4-3.

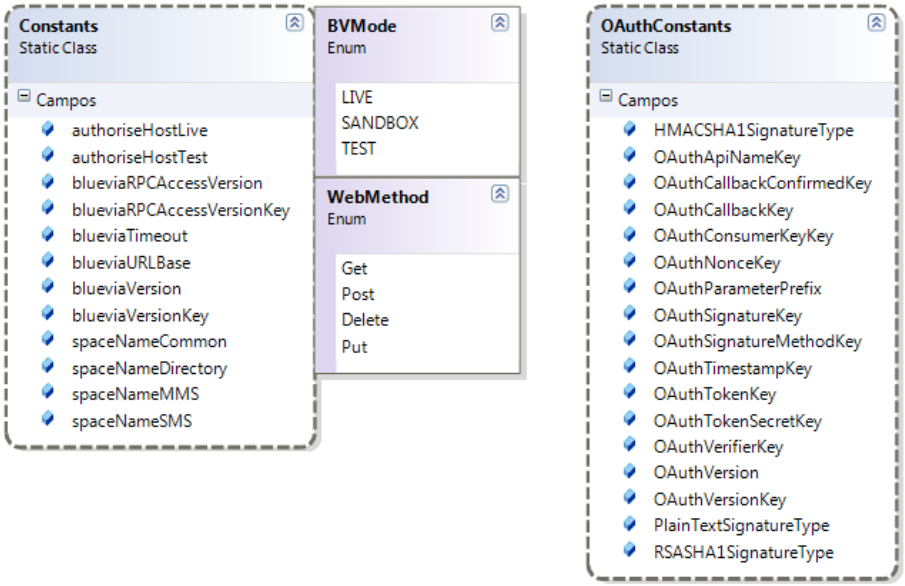


Figura 4-3 Diagrama de clases de constantes del nivel de acceso a la red

- **Constants** *(Bluevia.Core.Constants)*

  Esta clase estática define constantes usadas por toda la librería. Por un lado define elementos simples, como el *endpoint* de los servicios, o el *timeout* máximo que debe de agotar el conector antes de dar por perdida la respuesta; y por otro lado, define elementos enumerados:

  - **BVMode**. Que enumera los tres posibles modos de acceso a los servicios de *Bluevia*: *LIVE, SANDBOX, TEST.*
  - **WebMethod**. Que enumera las cuatro posibles operaciones HTTP, relativas a los métodos CRUD: *Get, Post, Delete y Put.*

- **OauthConstants** *(Bluevia.Core.Tools.OauthConstants)*

  Esta clase define los nombres de los campos que pueden componer la cabecera HTTP: *"Authorization: Oauth"* para el caso de *Bluevia*. También define algunos de los valores posibles, como la versión de *Oauth*, o las posibles codificaciones de firma.

  Esta clase se encuentra en el paquete *Tools* al ser un anexo directo de la clase ***OauthManager***.

Clases devueltas por el nivel de acceso a la red

Se muestra un esquema de estas, sus relaciones y métodos en la Figura 4-4.

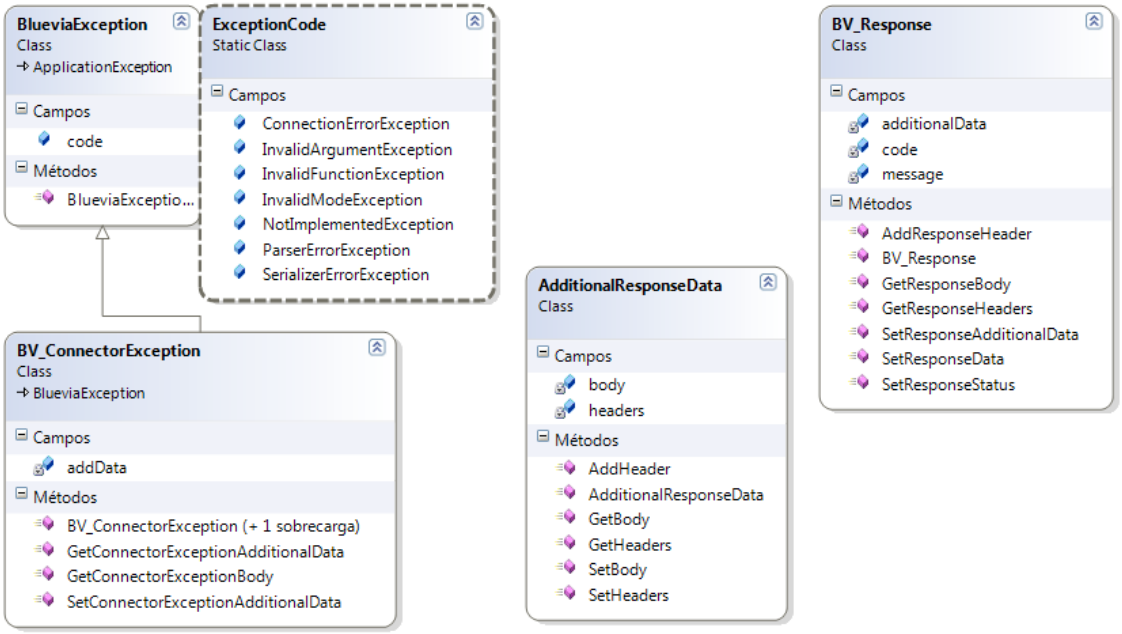


Figura 4-4 Diagrama de clases devueltas por el nivel de acceso a la red

- **BlueviaException** *(Bluevia.Core.Schemas.BlueviaException)*

  Esta es la excepción básica de *Bluevia*, que hereda directamente de la excepción de aplicación de ***C#***. Añade a aquella un campo *code* en el que poder guardar el código de error asociado a los especificados para los SDK. Dichos códigos se encuentran para el caso de ***.NET*** en la clase interna ***ExceptionCode***.

  Estas excepciones, surgen cuando la librería no puede solucionar el problema –recibidos datos incorrectos o falta de conectividad entre otros- por lo que no se capturarán dentro de la librería, y serán lanzadas directamente contra el usuario

- **ExceptionCode** *(Bluevia.Core.Schemas.BlueviaException -> ExceptionCode)*

  Esta clase estática e interna a la clase ***BlueviaException***, define los códigos de las posibles fuentes de excepcione contempladas en la librería.

- **BV_ConnectorException** *(Bluevia.Core.Schemas.BV_ConnectorException)*

  Esta clase, como se describe en b, surge cuando se recibe un error HTTP -40x o 50x-, y por tanto al poseer información equivalente a una respuesta HTTP, necesita de los mismos campos.

  Para ello, esta clase hereda de ***BlueviaException*** ampliando esta con un campo del tipo ***AdditionalResponseData*** para contener la información equivalente a una respuesta HTTP.

  La excepción es capturada durante la invocación de los métodos CRUD por la clase ***BV_BaseClient*** del nivel de inteligencia común, su contenido es procesado, y finalmente es traducida y lanzada hacia el usuario como ***BlueviaException***.

- **BV_Response** *(Bluevia.Core.Schemas.BV_Response)*

  Esta clase está diseñada para guardar las respuestas HTTP de los servicios de *Bluevia* –ver sección a del punto 3.3.1.2-, y para ello posee los campos:

  - `string` ***code*** y `string` ***message***, para guardar el código de la respuesta HTTP y su descripción.
  - `AdditionalResponseData` ***additionalData***, para guardar la "metainformación" y el posible cuerpo del mensaje.

  Estas respuestas se crearán una vez se haya verificado que la respuesta HTTP es válida, al finalizar el método *Call* del ***HTTPConnector***.

- **AdditionalResponseData** *(Bluevia.Core.Schemas.AdditionalResponseData)*

Esta clase modela un almacén en el que guardar la "metainformación", y el cuerpo de una respuesta HTTP. Y es usada con tal fin por las clases ***BV_Response*** y ***BV_ConnectorException***.

4.2.2.2 Clases del nivel de inteligencia común

Al igual que las clases del nivel de acceso a la red, estas clases también están contenidas en el paquete ***Core***. Desarrollan la funcionalidad común requerida por los servicios para acceder a *Bluevia*, con todas las opciones centralizadas en la clase ***BV_Client***.

a. Clientes comunes

Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-5.

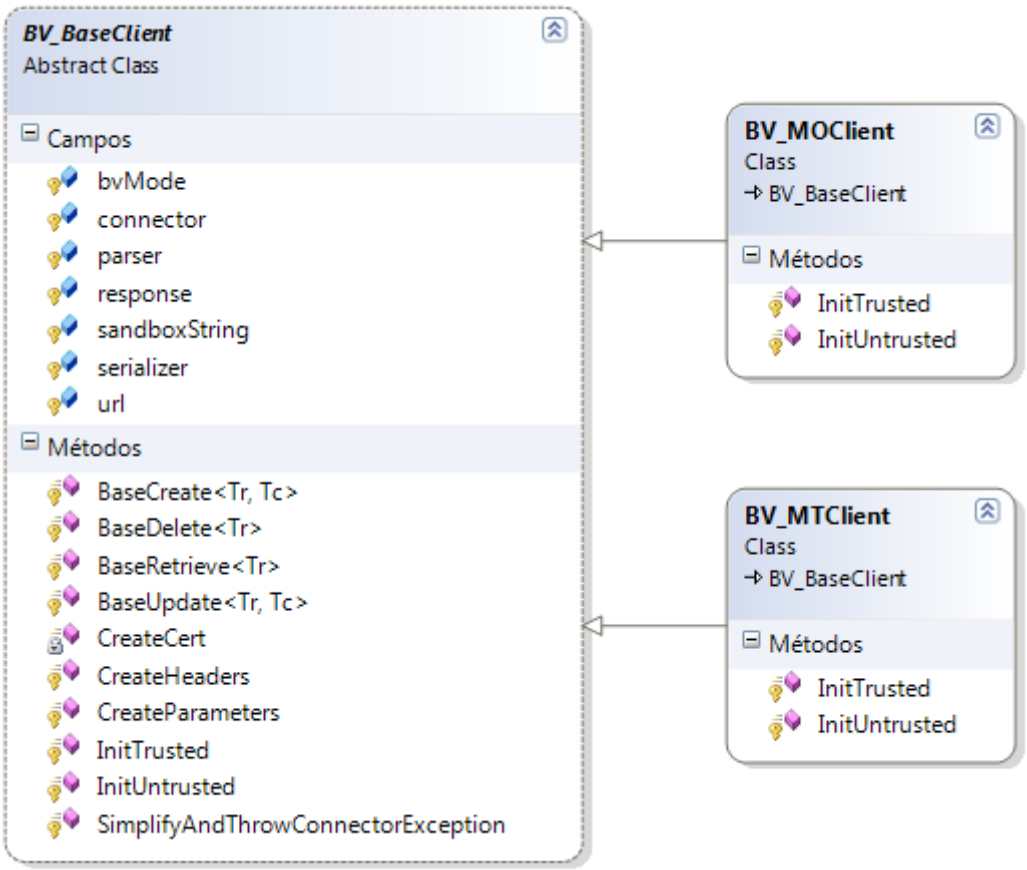


Figura 4-5 Diagrama de clientes base del nivel de inteligencia común

- **BaseClient** *(Bluevia.Core.Clients.BaseClient)*

  Esta es la clase abstracta principal de la que heredarán todos los clientes de API, y como se describe en "3.3.2.1: Cliente básico" engloba a toda la funcionalidad que puedan necesitar dichos clientes para acceder a los servicios de *Bluevia*. Para ello dispone de métodos y objetos para formatear el contenido de las peticiones, invocar las operaciones CRUD, y procesar el contenido de las respuestas; devolviendo luego el control al código específico del API.

A pesar de que no es instanciable, contiene métodos para configurar su funcionamiento a través de la iniciación de los siguientes campos:

- *BVMode* ***bvMode***. Este campo representa el modo operativo en el que se desea hacer funcionar el SDK –*LIVE*, *SANDBOX* o *TEST*- para que el cliente configure consecuentemente las URL de los servicios.

- *string* ***url***. En este campo se construye –de forma iterativa en los métodos "Init"- la dirección del API en uso, con el modo operativo apropiado. Lo utilizará cada servicio invocado, para crear la URL a la que enviar los mensajes.

- *BV_Connector* ***connector***. Es el conector de red que ofrece la funcionalidad de enviar mensajes HTTP con autenticación de *Bluevia.*

- *string* ***sandboxString***. Es un elemento de apoyo para crear inicialmente la URL, su valor viene determinado por el del modo operativo seleccionado.

Además de los anteriores, contiene campos que variarán con la invocación de cada servicio:

- *BV_Response* ***response***. En este objeto se guardará la respuesta recuperada tras cada invocación a un servicio de *Bluevia.*

- *Iparser* ***parser***. En este campo se establecerá el "parseador" que tenga que procesar el cuerpo de la respuesta, del servicio en curso.

- *Iserializer* ***serializer***. En este campo se establecerá el "serializador" que tenga que formatear el cuerpo de la petición al servicio en curso.

Para manejar la funcionalidad de este cliente abstracto, se ofrecen los siguientes métodos:

- *Métodos Init.* Estos métodos inicializan las variables que configuran el funcionamiento del cliente, de forma iterativa. En ellos se configura el modo operativo en el que funcionará el SDK, se crean las URL de los API en curso, y se establecen los datos de autorización de la aplicación.

  Estos métodos son reescritos por las clases hijas, para llamar primero al *Init* del padre, y luego realizar la configuración referente al nivel de la clase en cuestión. Son invocados desde los clientes finales de API.

- *Métodos BaseCRUD*. Estos métodos son los encargados de preparar el cuerpo del mensaje, enviarlo, y procesar de forma adecuada el cuerpo de la respuesta, según dicte el funcionamiento de su tipo de operación: *Create*, *Retrieve*, *Update*, *Delete*; usando para ello el "serializador", conector, y "parseador". En caso de que *Bluevia* devuelva un error, se invocará el método *SimplifyAndThrowConnectorException*, para empaquetar la respuesta HTTP y devolvérsela al usuario como excepción.

- *CreateParameters*. Este método será invocado por los clientes de API, para inicializar la lista de parámetros de URL requeridos en todas las peticiones a *Bluevia.*

- *CreateHeaders*. De forma equivalente al método anterior, este método es invocado por los clientes de API, para inicializar la lista de las cabeceras HTTP para la petición en curso.

Finalmente en esta clase existen los siguientes métodos de ámbito propio:

- *CreateCert*. Este método es invocado por *InitTrusted*, para crear un objeto contenedor de certificados SSL, con los certificados existentes en el sistema indicados por la ruta que el usuario pasa por parámetro durante la creación de un cliente de acceso privado; y dotar así de la conectividad necesaria al objeto ***BV_Connector***.
- *SimplyfyAndThrowConnectorException*. Este método es invocado por los *BaseCRUD*, cuando *Bluevia* responde con un error HTTP, para empaquetar la información recibida y devolvérsela al usuario como excepción.

- **BV_MOClient** *(Bluevia.Core.Clients.BV_MOClient)*

  Esta clase de es parte de la cadena de herencia, pensada para montar la estructura de la URL de los servicios de *Messagery*.

- **MT_Client** *(Bluevia.Core.Clients.MT_Client)*

  Esta clase de es parte de la cadena de herencia, pensada para montar la estructura de la URL de los servicios de *Messagery*.

b. Clases auxiliares

Las clases auxiliares en el nivel de lógica común son las encargadas de atender a los formatos para el cuerpo de los mensajes, encargándose de la "serialización" y el procesado de estos.

Para poder cumplir con el esquema de interfaces alcanzado en el diseño –ver punto 3.3.2-, en el que un mismo tipo de parámetros, serán considerados y tratados como diferentes tipos de objetos, se hará uso de la funcionalidad "*Reflections*"[52] proporcionada por el lenguaje ***C#***, para poder tratarlos en cierto modo como si fuesen tipos dinámicos.

"Serializadores"

Como se describe en el punto del diseño: 3.3.2.2, los "**Serializadores**" son los encargados de transformar la información que se vaya a enviar en el cuerpo de la petición -empaquetada inicialmente en una de las clases contenedoras-, al formato que requiera el servicio: XML, *FormURLEncoded o Multipart*.

Será el **Cliente de API** el que instancie uno o varios "serializadores" diferentes para cubrir sus necesidades. De manera que en cada método distinto de invocación de servicio, selecciona el apropiado para formatear la información.

Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-6.

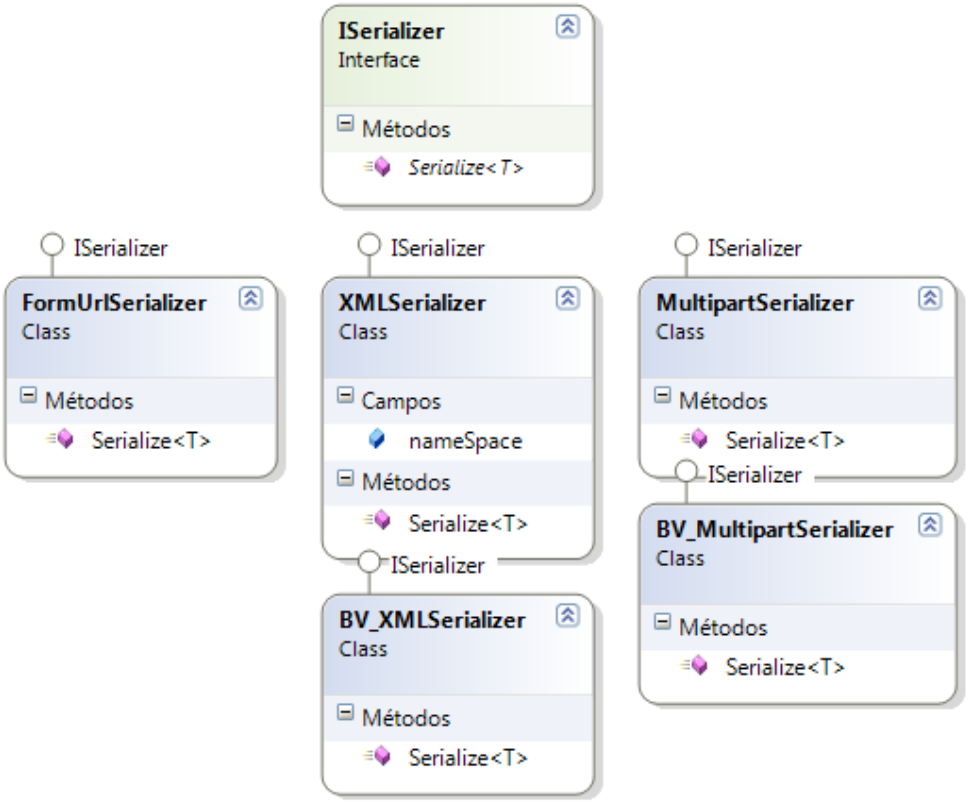


Figura 4-6 Diagrama de los serializadores

- **Iserializer** *(Bluevia.Core.Tools.Iserializer)*

  Esta interfaz define la funcionalidad de los "**Serializadores**" a través del método:

  - *Serialize<T>.* Este método recibe como parámetro un objeto contenedor de tipo genérico "T" llamado *entity* -será cada "serializador" el que defina el tipo de objeto que le hayan pasado-. Se creará entonces un bloque de caracteres con el contenido de los campos del objeto *entity*, adecuado al formato que especifique el "serializador", y finalmente se transformará el texto a un *array* de *bytes* para que pueda ser incrustado a posteriori en el cuerpo de la petición –método *Call* del ***HTTPConnector***-.

- **MultipartSerializer** *(Bluevia.Core.Tools.MultipartSerializer)*

  Este "serializador" se encarga de transformar listas genéricas de archivos adjuntos, en un archivo *Multipart* completo.

  - *Serialize<T>.* En primer lugar el método transforma el objeto "T" recibido –*entidad*- , en una lista de archivos adjuntos -definidos en el paquete de **MMS.Schemas** por coherencia con la estructura de funcionalidad, en detrimento de la ligereza de la carga de paquetes-. Puede verse el código de la transformación en la Figura 4-7.

```
List<AttachmentObject> attachmentList = entity as List<AttachmentObject>;
```

Figura 4-7 Fragmento de código que permite el *casting* de un objeto recibido sin tipo, al tipo lista de adjuntos.

Tras ello, por cada elemento de la lista, crea las cabeceras que lo describen; y añade, cabeceras y elemento, a un *stream* en memoria.

Tanto cabeceras como elementos, son escritos en el stream en formato byte, usando la codificación por defecto del sistema, para evitar errores de caracteres.

Todos los elementos son separados por un *boundary* temporal, del que finalmente se contarán las apariciones en el archivo *Multipart*; y en el caso de encontrarse más de lo debido –nº de adjuntos +1, es lo debido-, se realizará el proceso de montado de nuevo con un nuevo *boundary*.

Nota: Por esto último, el proceso de serializado de *Multiparts* es, probablemente, el proceso más pesado que pueda acontecer en la librería.

Esta clase no es usada directamente por los clientes para crear los *multiparts*, sino que es utilizada por la clase ***BV_MultipartSerializer*** para que le empaquete los archivos que crea.

- **BV_MultipartSerializer** *(Bluevia.Core.Tools.BV_MultipartSerializer)*

Dado que *Bluevia* requiere mensajes *Multipart* con un formato determinado, la "serialización" de este tipo se realiza en dos partes: La acometida por esta clase, ***BV_MultipartSerializer,*** que atañe al formato particular de *Bluevia*; y la de la clase ***MultiparSerializer*** que aborda la "serialización" general de mensajes *Multipart.*

- Método *Serialize<T>*. Se transforma el objeto "T" recibido –*entity*- en un objeto de información, adecuado al formato de *Multiparts* de *Bluevia*. Puede verse un fragmento del código de transformación en la Figura 4-8.

```
MultipartMessageType multipart = entity as MultipartMessageType;
```

Figura 4-8 Fragmento de código que permite el *casting* de un objeto recibido sin tipo, al tipo mensaje *Multipart*

- Como se verá más adelante, estos objetos contienen la información sobre varios archivos adjuntos, entre ellos, la de un mensaje corto tipo SMS.

  Con esta información, se crean los archivos adjuntos y se empaquetan en una lista homogénea, para invocar con ella el método *Serialize<T>* de la clase ***MultipartSerializer***, y sea esa clase la que cree el cuerpo del mensaje.

- **XMLSerializer** *(Bluevia.Core.Tools.XMLSerializer)*

De forma similar al caso *Multipart*, esta "serialización" se realiza en dos partes. Con la diferencia de que en el caso de XML, esta clase no es autosuficiente a la hora de crear los contenidos completamente formados, sino que necesita del ***BV_XMLSerializer*** para realizar el formato al completo. Esto es así, porque la parte del formato que atañe a *Bluevia,* son los espacios de nombres de los objetos "serializados"

-Namespaces-, información que forma parte del estándar, y que resulta imprescindible para invocar correctamente los servicios de *Bluevia*.

De modo que esta clase contiene:

- *System.Xml.Serialization.XmlSerializerNamespaces* ***namespace.*** Campo para que el *namespace* sea definido por alguna otra clase superior.

- Método *Serialize<T>.* Que utiliza el namespace, un *stream* de memoria y un elemento del *Framework* de ***.NET***, para crear el contenido. El código de la Figura 4-9 ilustra la instanciación de un "serializador" del *framework* adecuado a la entidad pasada.

```
using (System.IO.MemoryStream stream = new System.IO.MemoryStream())
            {
             System.Xml.Serialization.XmlSerializer serializer = new
System.Xml.Serialization.XmlSerializer(typeof(T));
             …
            }
```

Figura 4-9 Fragmento de código que permite preparar al serializador para aplicar el formato XML adecuado a un tipo T recibido

Al igual que en el caso de *Multipart*, esta no es la clase que será utilizada directamente por los clientes de *Bluevia* para crear los cuerpos de los mensajes; sino que será llamada la clase ***BV_XMLSerializer*** para realizar las acciones previas y finalmente invocar al ***XMLSerializer*** para generar el contenido.

- **BV_XMLSerializer** *(Bluevia.Core.Tools.BV_XMLSerializer)*

  Dado que *Bluevia* requiere mensajes *XML* con un *namespace* determinado, la "serialización" de este tipo se realiza en dos partes: La acometida por esta clase, ***BV_XMLSerializer***, que atañe a la generación de los *namespaces* de los objetos de *Bluevia que van a ser serializados*; y la de la clase ***XMLSerializer*** que aborda la "serialización" a XML del objeto.

  - *Serialize<T>*. En este caso el método crea un objeto *namespace*, con los espacios de nombres relativos al objeto "T" recibido:

    Para provisionarlo en el campo de un ***XMLSerializer*** y así poder ejecutar su método *Serialize.*

- **FormUrlSerializer** *(Bluevia.Core.Tools.FormUrlSerializer)*

  En el caso del formato *FormUrlEncoded*, *Bluevia* no especifica ninguna particularidad. Por lo que solo es necesario un "serializador" que se atenga a la especificación estándar.

  - *Serialize<T>*. Se transformará el objeto "T" a una colección de pares Campo-Valor, que serán unificados con el formato adecuado en una sola cadena. Se mustra el fragmento de código que realiza dicha acción en la Figura 4-10.

```
var body = string.Join("&", pairs.Select(q => q.Key + "=" +
Uri.EscapeDataString(q.Value)).ToArray());
```

Figura 4-10 Fragmento de código del FormUrlSerlializer para unificar parámetros en una sola cadena

"Parseadores"

Como se describe en el punto 3.3.2.3, los "**Parseadores**" son los encargados de transformar la información que vuelve encapsulada en el cuerpo de la respuesta, en forma de *array* de *bytes,* a un objeto contenedor, cuyos campos puedan ser accedidos fácilmente, para devolverle la respuesta al usuario.

Como en el caso de los serializadores, existen tres formatos a procesar: XML, *FormURLEncoded* o *Multipart*. Y es el **CLiente de API** el que instancie uno o varios "**Parseadores**" diferentes para cubrir sus necesidades. De manera que en cada método distinto de invocación de servicio, selecciona el apropiado para procesar la información. Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-11.

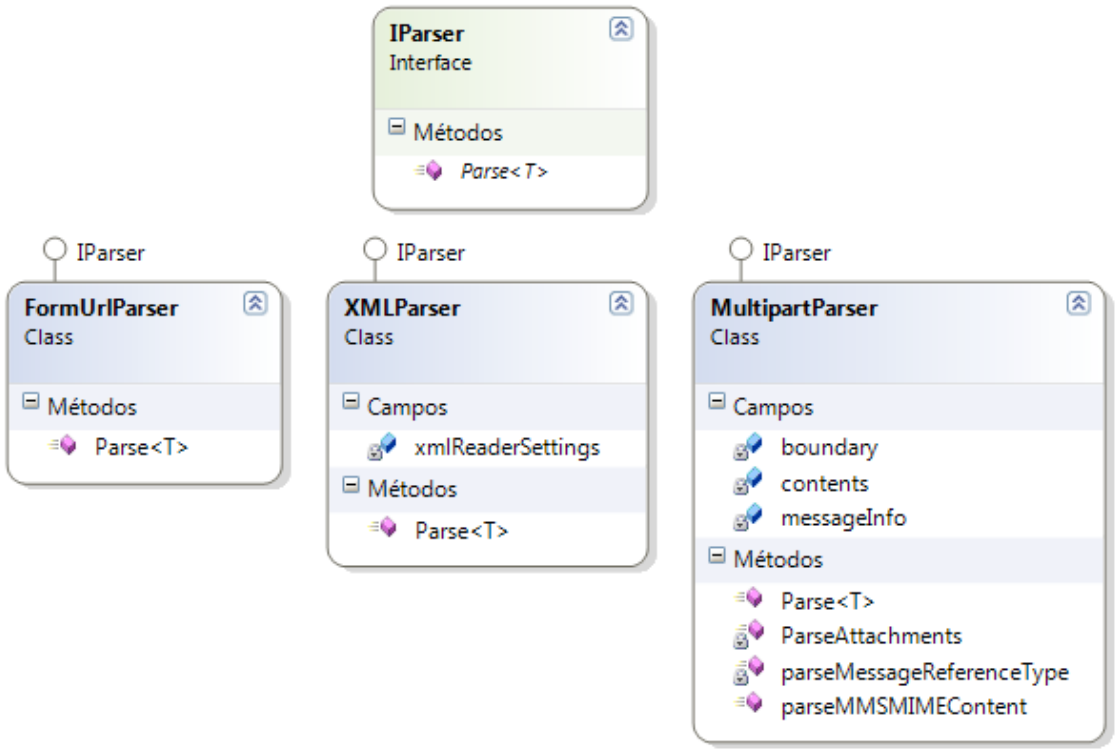


Figura 4-11 Diagrama de los "Parseadores"

- **Iparser** *(Bluevia.Core.Tools.Iparser)*

  Esta interfaz define la funcionalidad de los "parseadores" a través del método:

  - *Parse<T>*. Este método recibe un *array* de *bytes* –*stream*- con la respuesta en uno de los tres formatos posibles. Transforma este *array* a texto, como se muestra en el código de la Figura 4-12.

```
string str = Encoding.Default.GetString(stream);
```

Figura 4-12 Fragmento de código de los "Parseadores" que transforma el cuerpo del mensaje en un texto procesable

y lo procesa, para crear con la información recibida un objeto del tipo "T" -especificado durante la invocación del método-.

- **MultipartParser** *(Bluevia.Core.Tools.MultipartParser)*

  Este "parseador" se encarga de procesar el mensaje recibido, troceándolo en cada una de sus partes; y rellenar con ellas un objeto del tipo *MessageType* junto con una lista de *MIMEContents*, empaquetados en un objeto *MMSMessage.*

  - `string` ***boundary***, `List<Bluevia.Messagery.MMS.Schemas.MIMEContent>` ***contents,*** `MessageType` ***messageInfo.*** Son campos relativos al format Multipart.

  - *Parse<T>*. Las respuestas *Multipart* de *Bluevia*, están estructuradas en dos partes diferenciadas: Un objeto XML con la información equivalente a un SMS, y una lista de archivos adjuntos.

    Por lo que para que el proceso de procesado fuese claro y accesible, después de transformar los bytes a texto, este método inicia una cadena de invocaciones a otros métodos de cometido acotado*:*

    ParseMessageReferenceType->ParseMMSMIMEContent->ParseAttachments

  - *ParseMessageReferenceType*. Separa las dos partes del *Multipart*, y utiliza un ***XMLParser*** para procesar y reconstruir el SMS. Tras ello, pasa la segunda parte al método *ParseMMSMIMEContent.*

  - *ParseMMSMIMEContent*. Este método se encarga de procesar las cabeceras de la segunda parte del *Multipart* –los adjuntos-, y llegado el caso, recuperar el *boundary* interno. Una vez finalizados los preparativos, llama al método *ParseAttachments* para que se encargue de recuperar los archivos adjuntos, y crear con ellos una lista.

  - *ParseAttachments*. Este método trocea el bloque de adjuntos en sus partes individuales, y con cada una de ellas crea un objeto *MIMEContent,* rellenando sus campos, tanto con los bytes de información, como con las cabeceras que la describen.

  A diferencia de los "serializadores" o los demás "parseadores", esta clase está desarrollada para procesar únicamente los mensajes *multipart* del API de MMS, ya que el código busca expresamente la información de mensajería contenida.

- **XMLParser** *(Bluevia.Core.Tools.XMLParser)*

  Este "parseador" se encarga de construir y rellenar, con el mensaje recibido, el objeto contenedor de *Bluevia* especificado por "T", durante la invocación.

  Como en el caso del "serializador" de XML, se usará funcionalidad especifica del *Framework* de ***.NET***.

- *System.Xml.XmlReaderSettings* ***xmlReaderSettings***. Este es un objeto estático que especifica parámetros de configuración para el objeto del *Framework* que se encarga del procesado XML.

- *Parse<T>*. En este "parseador", este método tan solo invoca la capacidad de "deserialización" de una de las clases de ***.NET*** para generar el objeto especificado por "T". El código de la Figura 4-13 muestra cómo se "deserializa" el cuerpo, con un "serializador" del *framework.*

```
var xmlSerializer = new
System.Xml.Serialization.XmlSerializer(typeof(T));
var x = System.Xml.XmlReader.Create(new
System.IO.StringReader(str),xmlReaderSettings);
return (T)xmlSerializer.Deserialize(x);
```

Figura 4-13 Fragmento de código para usar las librerías de "deserialización" de *framework*

- **FromUrlParser** *(Bluevia.Core.Tools.FromUrlParser)*

  Esta clase transforma el contenido de la respuesta a pares Campo-Valor:

  - *Parse<T>*. Después de transformar los bytes, se trocea el texto en cada uno de los pares y se rellena con ellos una colección de pares.

4.2.2.3 Clases de inteligencia de API

En este nivel se encuentran las clases referentes a cada servicio, contenidas en su respectivo paquete: *Oauth, Advertising*, *Directory,* Location*, Messagery (SMS* y *MMS) y Payment.* Estas establecen los requisitos particulares para acceder a cada servicio, usando la funcionalidad de los otros dos niveles, y ofrecen una interfaz de uso de la librería de cara al exterior.

Dado que todos los paquetes de API tienen una estructura similar, y están compuestos por los mismos tipos de clases, se enumeran a continuación y se describe su funcionamiento genérico; dejando para los apartados específicos de cada API la descripción de las particularidades que puedan existir.

Para la descripción de estas clases genéricas, se usará la palabra "API" en vez del propio nombre del API en cuestión –así el equivalente a la clase genérica ***BV_APIClient*** para el API de *Advertising*, sería ***BV_AdvertisingClient***-.

Clientes de inteligencia de API

- **BV_APIClient** *(Bluevia.API.Client.BV_API_Client)*

  Estas clases serán los clientes comunes –ver punto 3.3.3.1- de cada API. En ellas está contenido el código de configuración de las clases de la librería, y de los procesos de invocación, para acceder a los servicios del API.

En caso de que los diferentes servicios del API necesiten variar entre diferentes tipos de "serializadores" o "parseadores", el cliente definirá un campo para cada uno de ellos, y así poder ir seleccionándolos según se invoque cada servicio.

- `Iserializer tipoDeSerializer.` "N serializadores" Si el cliente usa más de uno –o uno, y servicios que no lo necesiten-.

- `Iparser tipoDeParser.` "N parseadores" Si el cliente usa más de uno –o uno, y servicios que no lo necesiten-.

En caso de necesitar solo uno o ninguno de estos objetos, estos campos no serán necesarios, pues se habrá establecido una instancia fija en el campo correspondiente heredado del ***BV_BaseClient***.

Por otro lado, todos los clientes comunes ofrecerán los siguientes métodos:

- *InitUntrusted* e *InitTrusted*. Estos dos métodos protegidos, se encargan de preparar la librería para acceder a los servicios del API. El primero será invocado en la versión pública de la librería, y el segundo en la privada.

  En primer lugar, invocan al método *Init* de la clase padre –BV_BaseClient salvo los casos de mensajería-, y tras ello invoca al método *InitApiUrlAndObjects* para que complete la URL, e instancie los "parseadores" y "serializadores" necesarios para el API.

- *InitApiUrlAndObjects.* Estos métodos privados generarán la cadena de la URL que apunte al *endpoint* adecuado al API. En caso de que las peticiones a los servicios que ofrece, o sus respuestas, contengan un cuerpo, instanciarán los "serializadores" y "parseadores" adecuados para procesarlos.

- *ServiciosProcess*. Estos métodos existirán en caso de que un servicio ofrezca varios puntos de entrada –o servicios virtuales, ver tablas de 3.3.3.2 -. Son métodos protegidos, y serán invocados por los métodos "*Servicios"* –de esta misma clase, o de la del cliente final-.

  Concentrarán los pasos necesarios para formatear y enviar las peticiones a todas las opciones del servicio; y tras ello recuperar y procesar las respuestas para devolver la información al usuario.

  En primer lugar verificará que se han recibido todos los datos necesarios para invocar el servicio, y en algunos casos, completará dicha información.

  Tras ello, se inicializan los datos necesarios referentes a la petición, usando para ello los datos recibidos, y los específicos de cada servicio: Parámetros de URL, cabeceras de la petición y objetos del cuerpo.

  Finalmente invoca el método CRUD del ***BV_BaseClient*** y devuelve la respuesta recibida.

- *Servicios.* Estos métodos serán siempre los puntos de entrada para la invocación de un servicio.

Los parámetros recibidos serán los mismos –en número, nombre, tipo y obligatoriedad- para todos los SDK de los diferentes lenguajes; como consecuencia de la decisión de diseño de homogeneizar las interfaces.

En caso de que un servicio del API contenga varias opciones o servicios virtuales, estos métodos prepararán los datos para invocar dicha opción, y finalmente delegarán en el *ServicioProcess* relativo.

Si no hay varias opciones, estos métodos realizarán la función completa de un *ServicioProcess*, ya que ese método no existirá.

- ***BV_API*** *(Bluevia.API.Client.BV_API)* y ***BVT_API*** –para la versión privada-

  Serán los clientes finales que hagan de interfaz de cara al exterior.

  - Su constructor recibirá por parámetro los datos necesarios para:

    -Identificar a la aplicación que vaya a usar los servicios –*Consumers*-.

    -Definir el modo de funcionamiento –*LIVE, TEST* o *SANDBOX*-.

    -Si fuera necesario, identificar al usuario –*Tokens* en los clientes públicos -.

  Luego iniciará la cadena de invocaciones de los métodos *Init* –*InitUntrusted* en el los clientes ***BV_API***, e *InitTrusted* en los ***BVT_API***-.

  - *Servicios*. En el caso de que existan diferencias en la invocación de los servicios entre la versión privada y la pública, esta clase redefinirá los métodos de invocación de los servicios de *Bluevia* desarrollados en las clases ***BV_API_Client***, recibiendo los parámetros adecuados en cada caso.

Herramientas y clases auxiliares de API

- ***APISimplifiers*** *(Bluevia.API.Tools.APISimplifiers)*

  Estas clases estáticas contienen métodos para extraer la información de los objetos devueltos por las respuestas de *Bluevia*, y montar otros objetos más simples, que contengan solo la información que pueda interesarle al usuario final. Estos objetos generados, son los mismos para todas las versiones de los SDK de los diferentes lenguajes –mismos nombres, campos y tipos-, como consecuencia de la decisión de diseño de homogeneizar las interfaces.

  - *SimplifyResponseObjectType*. Serán los métodos que se encarguen de transformar una respuesta compleja del tipo "Response*ObjectType"* –de las generadas por los "parseadores"- en el objeto simple que necesita el usuario.

- ***APITools*** *(Bluevia.API.Tools.APITools)*

  Estas clases estáticas contienen métodos que puedan necesitar los clientes del API en cuestión –transformadores de códigos, montadores de objetos y otras herramientas-.

Objetos contenedores y constantes de API

- ***Schemas*** *(Bluevia.API.Schemas.Schemas)*

  Estas clases de definición de múltiples objetos, son autogeneradas con los documentos XSD creados por los diseñadores de *Bluevia*, para definir los nombres, formatos y campos de información que tienen que viajar en los mensajes del API; como puedan ser los errores devueltos, los tipos de identificadores de usuario o los elementos XML que viajan en los cuerpos entre otros

> **Nota: Dada la cantidad de objetos que definen estas clases** –que no han sido diseñadas ni implementadas en el ámbito de este proyecto-, **y dado en todos los paquetes se repiten muchos de los objetos definidos; estas clases no serán descritas, ni se mostrarán en los diagramas.**

.

- ***Constants*** *(Bluevia.API.Costants)*

  Al igual que en el caso de las clases ***Constants*** del nivel de acceso a la red, las del nivel de inteligencia de API definirán los tipos enumerados y las constantes que serán usadas por las clases del paquete; por ejemplo los *endpoints* del respectivo API y de sus servicios.

a. Oauth

Este paquete contiene los clientes y objetos necesarios para invocar los servicios de autorización de aplicaciones –y de transacciones, pero esta funcionalidad será expuesta más adelante en el API de *Payment*- de *Bluevia*, y obtener de ellos los *Tokens*.

Además de lo descrito en la implementación de las clases de inteligencia de un API genérico, el paquete de *Oauth* particulariza en lo siguiente:

Dado que el servicio de *Oauth* para autorizar aplicaciones mediante *Tokens,* solo se usa en las que usan servicios públicos *–ya que se usan certificados SSL en el caso de los privados-*, este API tan solo contiene código para dichos servicios.

Clientes de Oauth

Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-14

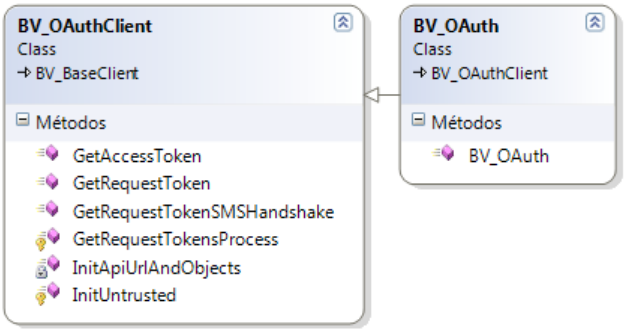


Figura 4-14 Diagrama de clientes de *Oauth*

- **BV_OauthClient** *(Bluevia.Oauth.Client.BV_OauthClient)*

  Este cliente usa la operación *BaseCreate*, y tanto el "serializador" como el "parseador" *FormURL* para todas sus peticiones a los servicios.

  Dado que Oauth carece de funcionalidad privada, en este cliente no extistirá el método *InitTrusted.*

  - *GetRequestToken.* Será la entrada virtual para recuperar los *Tokens* temporales de forma normal. Tan solo verifica que no se le pasen los parámetros especiales que pertenecen a las otras entradas virtuales, e invoca al *GetRequestTokenProcess.*
  - *GetRequestTokenSMSHandshake.* Otra entrada virtual para recuperar los *Tokens* temporales; en la que se verifica que la dirección *callback* en la que se quiere recibir el verificador es numérica, y se invoca a *GetRequestTokenProcess*.
  - *GetRequestTokenProcess.* Es el proceso completo que engloba la funcionalidad para recuperar los *Tokens* temporales, para todos los casos posibles. Como particularidad sobre un "*Proccess"* genérico, este método añadirá claves especiales a los parámetros de la cabecera de HTTP: Authorization: Oauth.
  - *GetAccessToken.* Este método usará los *Tokens* temporales y el validador, para recuperar los *Tokens* que finalmente autoricen a una aplicación o transacción. Como particularidad especial, en esta operación se añadirá la clave "*OauthVerifier"* a los parámetros de la cabecera de HTTP: Authorization: Oauth.

- **BV_Oauth** *(Bluevia.Oauth.Client.BV_Oauth)*

  Como este API está pensado para autorizar el uso de los servicios y transacciones públicos, el constructor de esta clase no exige que se le pasen los *Tokens*, ya que será este API quien se encargue de conseguirlos.

Herramientas y clases auxiliares de Oauth

Se muestra un esquema de estas y métodos en la Figura 4-15.

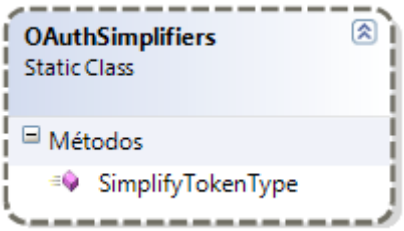


Figura 4-15 Diagrama del simplificador de *Oauth*

- **OauthSimplifier** *(Bluevia.Oauth.Tools.OauthSimplifier)*

  Leer comportamiento genérico de los simplificadores de objetos en: "Herramientas y clases auxiliares de API".

  - *SimplifyTokenType.* Este método encapsula en un objeto ***RequestToken*** la información de la respuesta en formato "Diccionario de pares".

Objetos contenedores y constantes de Oauth

Este paquete carece de clase ***Schemas***. Se muestra un esquema de estos, sus relaciones y campos en la Figura 4-16.

Por otro lado los objetos *Token*, a pesar de ser relativos al API de *Oauth,* dependen del paquete Core; ya que serán utilizados también por el API de *Payment* durante las autorizaciones de sus transacciones. Además, es probable que, siendo este un elemento tan ligado a la base del funcionamiento de *Bluevia*, pueda ser utilizado en futuras funcionalidades.

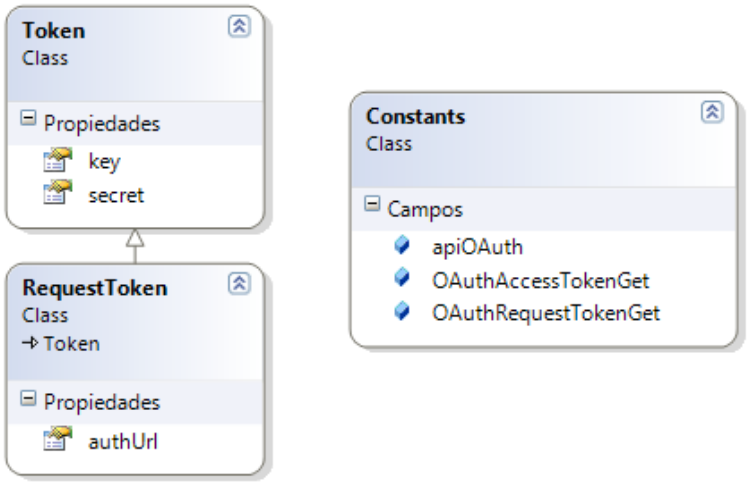


Figura 4-16 Diagrama de las clases contenedoras y constantes *de Oauth*

- **Token** *(Bluevia.Core.Schemas.Token)*

  Como se explica en 2.1.3, los *Token Credentials* son un par de claves alfanuméricas –pública y privada-, que define el protocolo Oauth, y emplea *Bluevia* para identificar que un usuario ha autorizado a una aplicación a realizar operaciones en su nombre.

  De modo que esta clase será utilizada como objeto respuesta de los servicios de autorización de aplicaciones u operaciones.

- **RequestToken** *(Bluevia.Core.Schemas.RequestToken)*

  Esta clase extiende al *Token* añadiéndole una URL –generada según el modo de acceso seleccionado-, en la que el usuario final deberá autorizar a la aplicación, para que esta pueda acceder a los servicios deseados.

- **Constants** *(Bluevia.Oauth.Constants)*

  Leer la funcionalidad genérica de las clases de constantes en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

b. Advertising

Este paquete contiene los clientes y objetos necesarios para poder invocar los servicios de recuperación de anuncios.

Además de lo descrito en la implementación de las clases de inteligencia de un API genérico, el paquete de *Advertising* particulariza en lo siguiente:

Clientes de Advertising

Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-17.

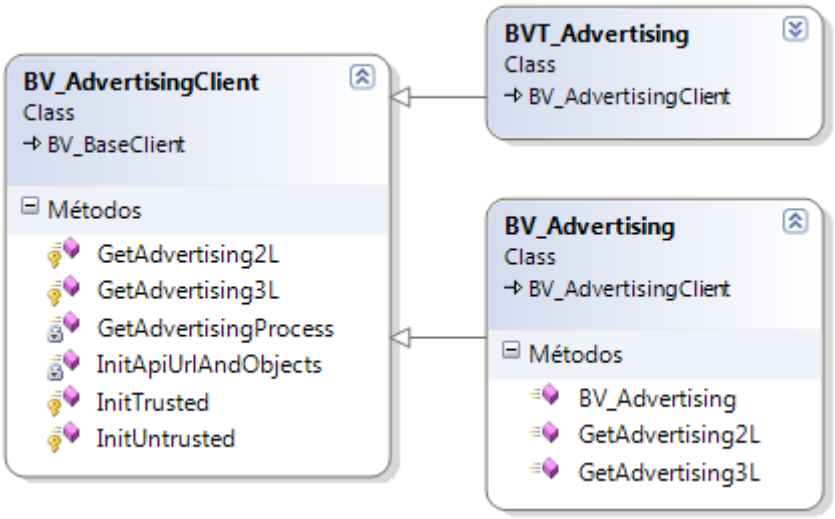


Figura 4-17 Diagrama de clientes de *Advertising*

- **BV_AdvertisingClient** *(Bluevia.Advertising.Client.BV_AdvertisingClient)*

  Este cliente usa la operación *BaseCreate*, un "serializador" *FormURL y un "parseador" XML.*

  Dado que en este API existen diferencias entre las versiones privadas y públicas de los servicios, los métodos de este cliente se protegen, para que los puntos de entrada se encuentren en los clientes finales.

  - *GetAdvertising2L*. Este método es una entrada virtual para recuperar anuncios, sin la autorización del usuario. Verifica que todos los datos que particularizan este modo del servicio han sido proporcionados, e invoca a *GetAdvertisingProccess.*
  - *GetAdvertising3L*. Este método es la segunda entrada virtual para recuperar anuncios, donde será necesaria la autorización del usuario para funcionar.

Verifica que esta autorización ha sido proporcionada chequeando la existencia de *Tokens* en la librería, y tras ello invoca a *GetAdvertisingProccess.*

- *GetAdvertisingProccess.* Contiene particularidades para el caso privado, por lo demás es como la genérica.

- **BV_Advertising** *(Bluevia.Advertising.Client.BV_Advertising)*

Dado que se permiten recuperar anuncios sin la autorización del usuario, el constructor de esta clase no necesita que se le pasen los *Tokens.*

Por otro lado, los métodos de acceso a los servicios virtuales del cliente común, son reescritos en esta clase –y en ***BVT_Advertising***- para definir en estos las diferencias entre servicio público y privado.

- **BVT_Advertising** *(Bluevia.Advertising.Client.BVT_Advertising)*

Herramientas y clases auxiliares de Advertising

Se muestra un esquema de estos y sus métodos en la Figura 4-18.

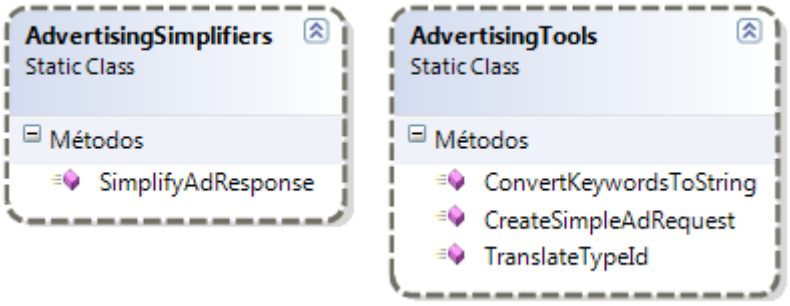


Figura 4-18 Diagrama de clases auxiliares de *Advertising*

- **AdvertisingSimplifiers** (Bluevia.Advertising.Tools.AdvertisingSimplifiers)

Leer comportamiento genérico de los simplificadores de objetos en: "Herramientas y clases auxiliares de API".

- **AdvertisingTools** *(Bluevia.Advertising.Tools.AdvertisingTools)*

Para facilitar la invocación del servicio, se pedirá al usuario que aporte la información de invocación en forma de parámetros simples. De modo que será esta clase la que se encargue de organizar y completar esta información para poder realizar correctamente las peticiones.

- *ConvertKeywordsToString.* Este método convierte el *array* de *strings* de las palabras clave de petición del anuncio, en una cadena convenientemente formateada.

- *TranslateTypeId.* Dado que el servicio de anuncios de *Bluevia* distingue entre imagen y texto por una clave numérica, se ofrece al usuario un objeto enumerado con ambas opciones; y este método traduce lo seleccionado a la clave adecuada que entienda *Bluevia.*

- *CreateSimpleAdRequest.* Este método monta un objeto ***SimpleAdRequest*** de los especificados en los ***Schemas*** de *Bluevia* –con la forma necesaria para realizar la invocación del servicio-, a partir de los parámetros individuales que pasase el usuario.

Objetos contenedores y constants de Advertising

Se muestra un esquema de estos, sus relaciones y campos en la Figura 4-19.

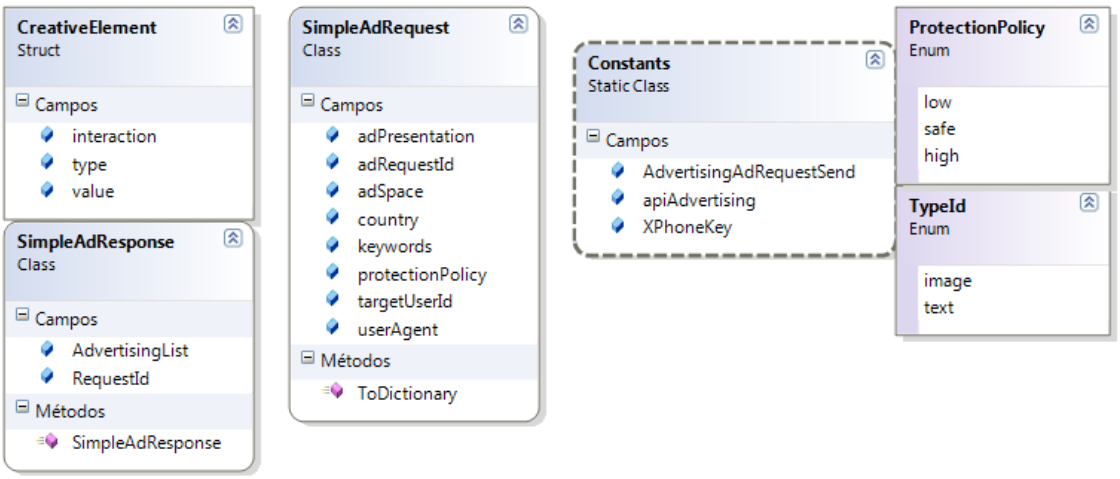


Figura 4-19 **Diagrama de las clases contenedoras y constantes *de Advertising***

- **SimpleAdRequest** *(Bluevia.Advertising.Schemas.SimpleAdRequest)*

  Esta clase define los campos de datos existentes que definen el anuncio a recuperar y contiene el método:

  - *ToDictionary,* que crea un "diccionario de pares Campo-Valor", para que pueda ser apropiadamente "serializado".

- **SimpleAdResponse** *(Bluevia.Advertising.Schemas.SimpleAdResponse)*

  Esta clase contiene los campos de la respuesta de *Bluevia,* útiles para el usuario.

  - `string RequestId,` Un identificador de la petición realizada

- `CreativeElement[] AdvertisingList,` *array* de los anuncios recuperados.

- **CreativeElement** *(Bluevia.Advertising.Schemas.CreativeElement)*

  Este objeto contiene la información de un anuncio:

  - `TypeId type,` El tipo de anuncio, imagen o texto.
  - `string interaction,` Un enlace URL para redirigir, en caso de pulsar el anuncio.
  - `string value,` La parte visible del anuncio, en forma de texto, o en forma de URL con la dirección de la imagen.

- **Schemas** *(Bluevia.Advertising.Schemas.Schemas)*

  Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

- **Constants** *(Bluevia.Advertising.Constants)*

  Además de las constantes, declara dos tipos enumerados:

  - **ProtectionPolicy**. Que permite al usuario definir la directiva de madurez deseada para el anuncio.
  - **TypeId**. Que perite decidir entre un anuncio en imagen, o uno en texto.

c. Directory

Este paquete contiene los clientes y objetos necesarios para invocar los servicios de recuperación de información de directorio.

Además de lo descrito en la implementación de las clases de inteligencia de un API genérico, el paquete de *Directory* particulariza en lo siguiente:

Clientes de Directory

Se muestra un esquema de estos, sus relaciones y métodos en la Figura 4-20.

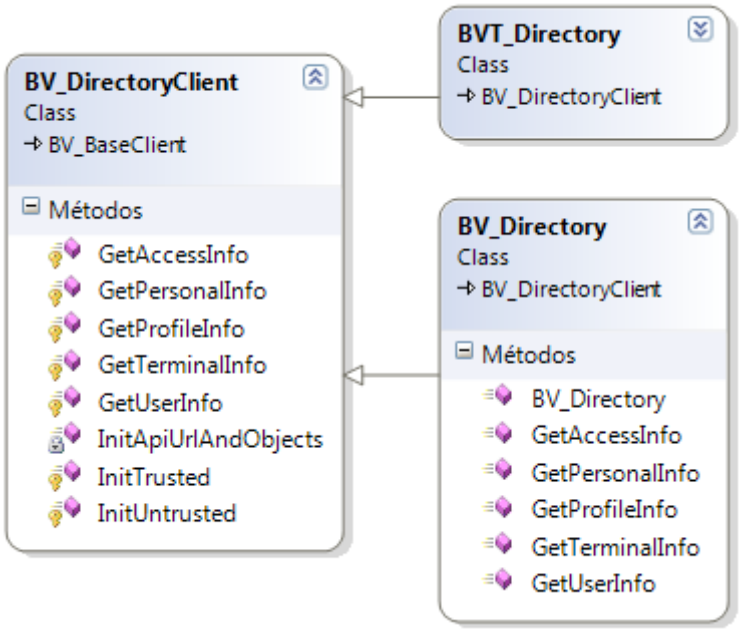


Figura 4-20 Diagrama de clientes de *Directory*

- **BV_DirectoryClient** *(Bluevia.Directory.Client.BV_DirectoryClient)*

  Este cliente usa la operación *BaseRetrieve* para invocar todos los servicios del API. No usa "serializador" –ya que POST no tiene cuerpo-, enviando los datos como parámetros de la URL. Utiliza para todas las respuestas un "parseador" XML.

  Dado que en este API existen diferencias entre las versiones privadas y públicas de los servicios, los métodos de este cliente se protegen, para que los puntos de entrada se encuentren en los clientes finales.

  Además, en este API, las URL se montan de forma particular, anteponiendo un identificador de usuario al servicio deseado. Para ello, a la hora de invocar a las operaciones CRUD, se llamará al método *CreateDirectoryServiceURL* de la clase ***DirectoryTools***.

  - *GetUserInfo*. Este método ofrece la funcionalidad de recuperar dos o más sets de datos completos. Si tan solo se quiere recuperar la información de uno, tendrá que invocarse el método correspondiente a ese set de datos.

    Para definir los sets deseados, hay que pasarle por parámetro un *array* de enumerados ***DirectoryDataSets*** con la selección realizada. En caso de querer recuperar los cuatro disponibles, se prescindirá del *array*.

  - *GetAccesInfo, GetPersonalInfo, GetProfileInfo, GetTerminalInfo.* Estos métodos ofrecen la funcionalidad de recuperar cada set de datos al completo, o solo algunos campos del set.

    Para definir los campos deseados, hay que pasarle por parámetro un *array* de enumerados ***SetFields*** con la selección realizada. En caso de querer recuperar todos los campos disponibles del set de información, se prescindirá del *array*.

- **BV_Directory** *(Bluevia.Directory.Client.BV_Directory)*

  Se reescriben en este cliente final, los métodos de acceso a los servicios del API, para definir los datos de identificación de usuario que diferencian entre acceso público –en el caso de esta clase- y privado –en caso de ***BVT_Directory***-.

- **BVT_Directory** *(Bluevia.Directory.Client.BVT_Directory)*

Herramientas y clases auxiliares

Se muestra un esquema de estas y sus métodos en la Figura 4-21.

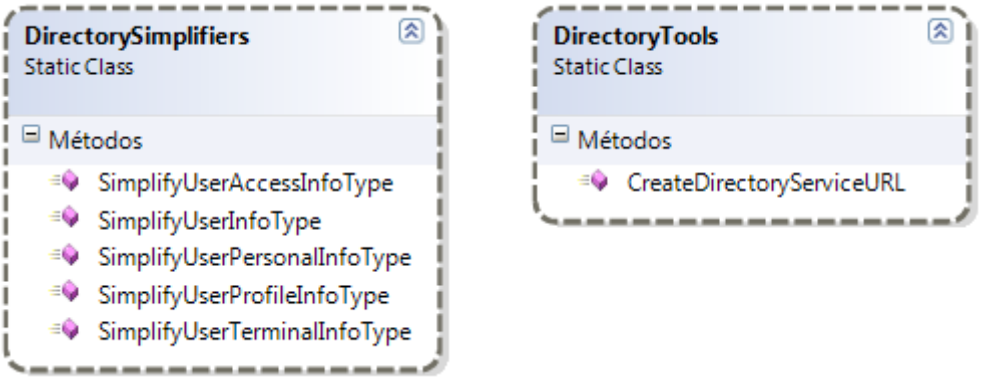


Figura 4-21 Diagrama de clases auxiliares de *Directory*

- **DirectorySimplifiers** *(Bluevia.Directory.Tools.DirectorySimplifiers)*

  Esta clase contendrá los métodos que monten los objetos que vayan a pasarse como respuesta al usuario. Filtra los campos recuperados, a los estrictamente especificados públicamente por *Bluevia* para cada servicio, convirtiéndolos en el proceso a tipos *string*.

- **DirectoryTools** *(Bluevia.Directory.Tools.DirectoryTools)*

  - *CreateDirectoryServiceUrl.* Dado que en el caso del API de *Directory* las URL de los servicios varían en su estructura con respecto a los demás API, este método se encarga de montar la última parte, anteponiendo un identificador de usuario al servicio, para que los procesos del resto del paquete mantengan la misma estructura que los demás.

Objetos contenedores y constantes de *Directory*

Se muestra un esquema de estos y sus campos en la Figura 4-22.

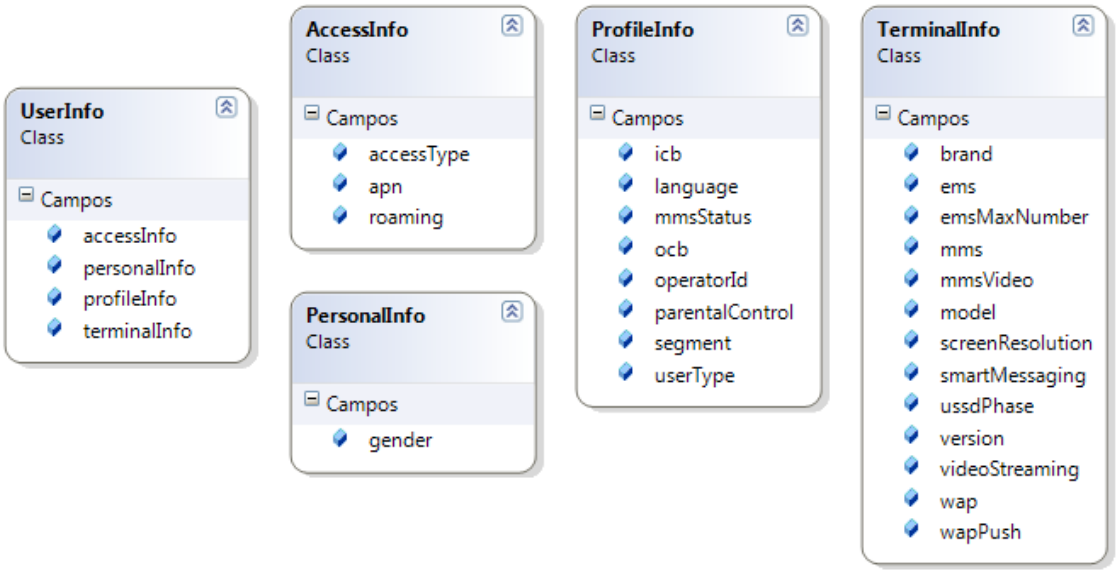


Figura 4-22 Diagrama de objetos contenedores de *Directory*

- **UserInfo** *(Bluevia.Directory.Schemas.UserInfo)*

  Este objeto se devolverá al usuario como respuesta de la operación *GetUserInfo*. Contiene los cuatro sets de datos disponibles, de los que sólo serán diferentes de *null* los que se hayan indicado en la petición –o todos, si se dejó en blanco la selección-.

- **AccessInfo** *(Bluevia.Directory.Schemas.AccessInfo)*, **ProfileInfo** *(Bluevia.Directory.Schemas.ProfileInfo)*, **PersonalInfo** *(Bluevia.Directory.Schemas.PersonalInfo)*, **TerminalInfo** *(Bluevia.Directory.Schemas.TerminalInfo).*

  Estos objetos serán devueltos al usuario como respuestas a las invocaciones respectivas a cada servicio de *Directory*. Cada uno contiene los campos –en tipo *string*- *Bluevia* define que debe contener. Como en el caso del ***UserInfo*** sólo serán diferentes de *null* los que se hayan indicado en la petición –o todos, si se dejó en blanco la selección-.

- **Schemas** *(Bluevia.Directory.Schemas.Schemas)*

Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3. Se muestra un esquema de estas y sus campos en la Figura 4-23.

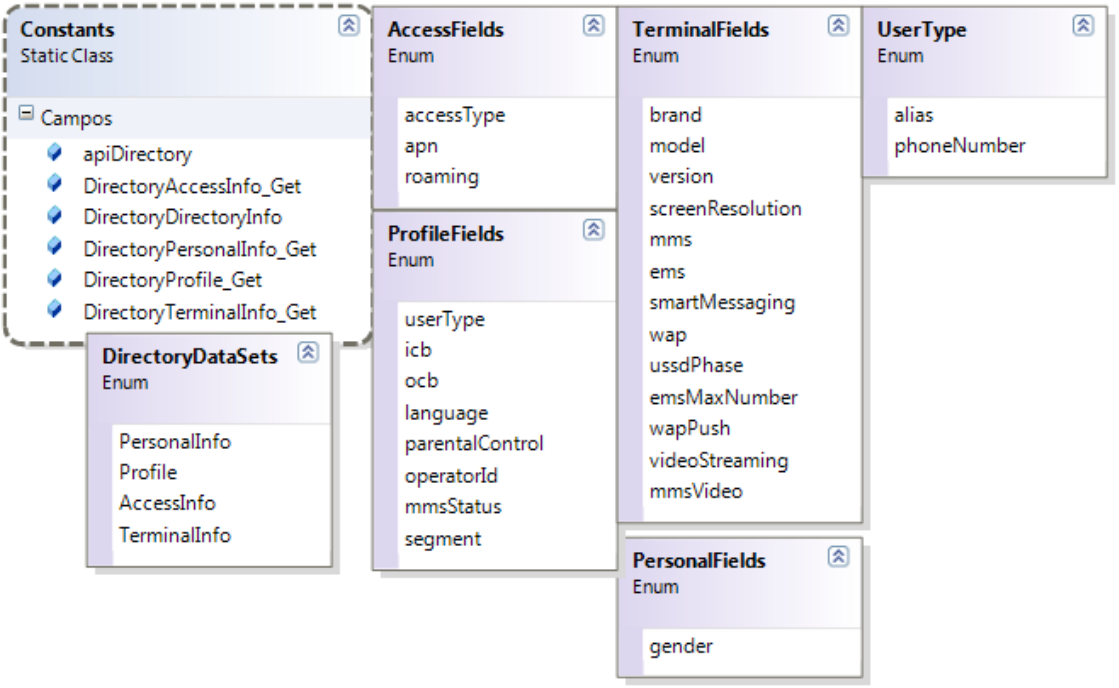


Figura 4-23 Diagrama de constantes y enumerados de *Directory*

- **Constants** *(Bluevia.Directory.Constants)*

  Además de definir las constantes, declara varios tipos de enumerados:

  - **DirectoryDataSets**, **AccessFields**, **ProfileFields**, **TerminalFields**, **PersonalFields.**

    Estos enumerados, permitirán indicar a los servicios de *Directory* que sets o campos de datos se desean recuperar. Usándolos de la forma descrita en el apartado del ***BV_DirectoryClient***.

  - **UserType.**

    Este enumerado permite definir el tipo de usuario del que se quiere recuperar la información de directorio.

d. Location

Este paquete contiene los clientes y objetos necesarios para invocar el servicio de localización por redes de telefonía móvil.

Además de lo descrito en la implementación de las clases de inteligencia de un API genérico, el paquete de *Location* particulariza en lo siguiente:

Clientes de Location

Se muestra un esquema de estos, sus relaciones y sus métodos en la Figura 4-24.

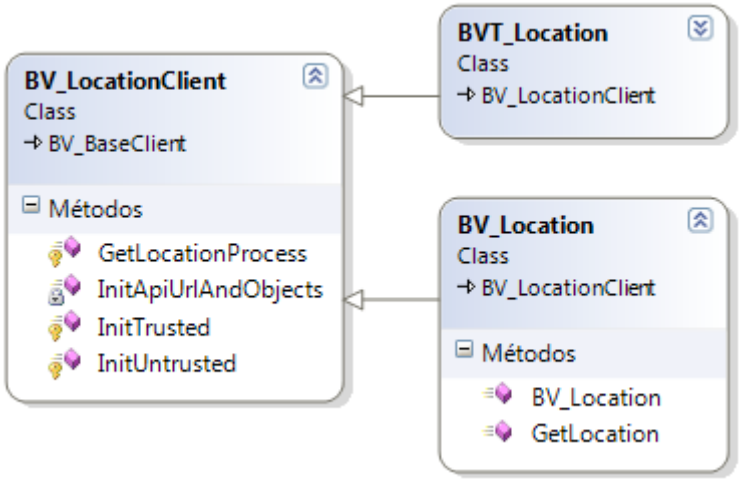


Figura 4-24 Diagrama de clientes de *Location*

- **BV_LocationClient** *(Bluevia.Location.Client.BV_LocationClient)*

  Este cliente usa la operación *BaseRetrieve* para invocar un único servicio. No usa "serializador" –ya que POST no tiene cuerpo-, enviando los datos como parámetros de la URL y utiliza un "parseador" XML.

  Dado que en este API existen diferencias entre la versión pública y privada del servicio, este cliente sólo contiene el método "*Process",* situándose los métodos de entrada en los clientes finales.

  - *GetLocationProcess.* En este caso, el método no engloba la funcionalidad de varios servicios virtuales, sino que realiza el proceso de invocación del servicio de recuperación de la localización de un terminal, para las versiones pública y privada.

- **BV_Location** *(Bluevia.Location.Client.BV_Location)*

  La entrada al único servicio que ofrece el API, se sitúa en este nivel, escribiéndose la funcionalidad relativa al acceso público, en esta clase.

  - *GetLocation.* Aquí se completan los datos de identificación de usuario, referentes al acceso público al servicio, y se invoca el *GetLocationProcess* de ***BV_LocationClient*** para completar el proceso de recuperación de la localización del terminal.

- **BVT_Location** *(Bluevia.Location.Client.BVT_Location)*

Herramientas y clases auxiliares

Se muestra un esquema de estas y sus métodos en la Figura 4-25.

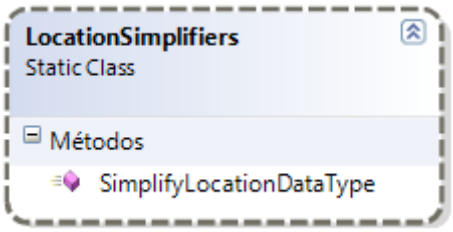


Figura 4-25 Diagrama del simplificador de *Location*

- **LocationSimplifiers** *(Bluevia.Location.Tools.LocationSimplifiers)*
  - *SimplifyLocationDataType.* Este método convierte varios de los campos de la respuesta recibida a *string*. Montando así un objeto ***LocationInfo*** que pueda ser devuelto al usuario.

Objetos contenedores

Se muestra un esquema de estos y sus campos en la Figura 4-26.

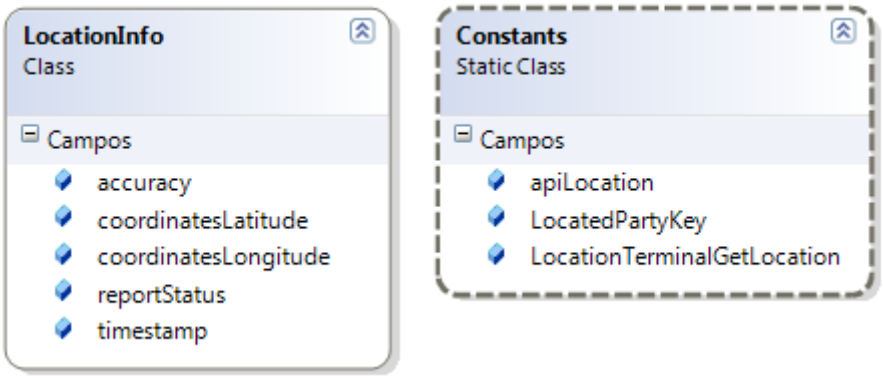


Figura 4-26 Diagrama de las clases contenedoras y constantes *de Location*

- **LocationInfo** *(Bluevia.Location.Schemas.LocationInfo)*

  Es el objeto devuelto al usuario, con la información de sus campos convertida a *string.*

- **Schemas** *(Bluevia.Location.Schemas.Schemas)*

  Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

- **Constants** *(Bluevia.Location.Constants)*

  Leer la funcionalidad genérica de las clases de constantes en la sección "Objetos contenedores y constantes de API" del punto 4.2.2.3.

e. Messagery (SMS y MMS)

Este paquete contiene los clientes y objetos necesarios para cubrir los aspectos de mensajería que ofrece *Bluevia.*

Como ya se vio en el punto 3.4.1.1 "Herencia en los módulos de inteligencia de *Bluevia*", el API de Mensajería contiene dos divisiones SMS/MMS y Envío/Recepción, lo que creará cuatro grupos.

Bluevia mantiene separados los *endpoints* de envío y recepción –ya que ambos tipos de servicios además de ser funcionalmente muy diferentes, tienen diferentes requisitos de autorización-; pero define los objetos como dependientes de SMS o de MMS, por lo que los clientes y objetos se agruparán en un paquete para cada tecnología.

Clientes de Messagery

SMS

Se muestra un esquema de estos, sus relaciones y sus métodos en la Figura 4-27.

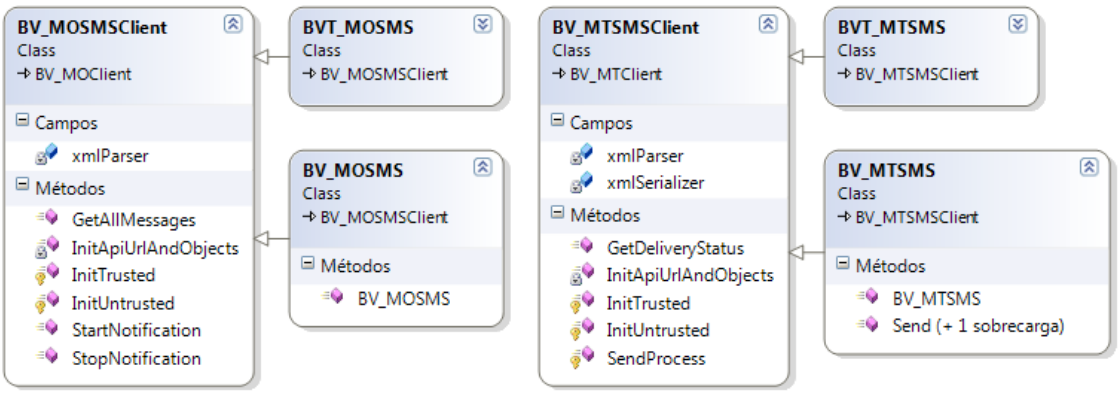


Figura 4-27 Diagrama de clientes de Mensajería SMS

MOBILE TERMINATED

Estos clientes contienen a funcionalidad ligada a los procesos de envío de SMS a terminales móviles.

- **BV_MTSMSClient** *(Bluevia.Messagery.SMS.Client.BV_MTSMSClient)*

Este es el cliente común de los servicios *Mobile Terminated* para mensajería SMS.

Como las algunos de los servicios necesitan "serializador" y "parseador" y otros no, se establece los campos *xmlSerializer* y *xmlParser*, para poder conmutar entre ellos y *null.*

- *SendProcess.* En este caso, el método no engloba la funcionalidad de varios servicios virtuales, sino que realiza el proceso de envío de un SMS para las versiones pública y privada del API.

  Será en los clientes finales, donde se sitúe el punto de entrada a este proceso.

  Usa la operación CRUD *BaseCreate*, y un "serializador" XML, para preparar la información del SMS a enviar.

  La particularidad de este método es la forma de recuperar la información de la respuesta; ya que esta no viaja en el cuerpo del mensaje, sino en una de las cabeceras. Por ello, este método no usa "parseador", y directamente saca la información del objeto ***BV_Presponse*** que le lega el ***BaseClient***.

- *GetDeliveryStatus.* Este método recupera el estado de un mensaje previamente enviado.

  Usa la operación *BaseRetrieve,* a la URL que fue devuelta como respuesta al envío del SMS.

  Finalmente utiliza un "parseador" XML, para procesar la información del estado del envió.

- **BV_MTSMS** *(Bluevia.Messagery.SMS.Client.BV_MTSMS)*

  La entrada al *SendProcess* del ***BV_SMSMTClient*** se sitúa en este nivel; escribiéndose la funcionalidad relativa al acceso público en esta clase.

  - *Send.* Aquí se completan los datos de identificación de usuario, referentes al acceso público al servicio, y se invoca al *SendProcess* de ***BV_SMSMTClient*** para completar el proceso de envío de SMS.

- **BVT_MTSMS** *(Bluevia.Messagery.SMS.Client.BVT_MTSMS)*

MOBILE ORIGINATED

Estos clientes contienen a funcionalidad ligada a los procesos de recepción de SMS originados por dispositivos móviles, y depositados en un buzón.

Como particularidad importante, hay que señalar que estos servicios no usan autorización de usuario en su versión pública, por los que los clientes no requerirán *Tokens.*

- **BV_MOSMSClient** *(Bluevia.Messagery.SMS.Client.BV_MOSMSClient)*

Este es el cliente común de los servicios *Mobile Originated* para mensajería SMS.

Cada uno de los servicios que invoca este cliente, funciona de forma diferente, por lo que los métodos, a diferencia de otros API, no usan los mismos elementos u operaciones.

- *GetAllMessages.* Este método descargará del buzón –definido por el parámetro *registrationId-* todos los SMS recibidos. Para ello usa la operación *BaseRetrieve* y procesa la respuesta con un "parseador" XML. Devolviendo finalmente un *array* de objetos ***SMSMessage***.

- *StartNotification.* Este método configura el buzón, para que reenvíe a un servidor –descrito por los parámetros que se envíen-, los SMS que le vayan llegando. Usa la operación *BaseCreate,* usando un "serializador" XML, para crear el cuerpo.

- *StopNotification.* Este método configura el buzón para parar el envío de SMS al servidor previamente definido. Usa la operación *BaseDelete*, por lo que no hay cuerpo que "serializar"; y tampoco necesita un "parseador" porque no hay respuesta –siempre que no haya error en el proceso-.

- **BV_MOSMS** *(Bluevia.Messagery.SMS.Client.BV_MOSMS)*

  Este cliente final, tan solo elimina de su construcción los parámetros *Tokens* de identificación de usuario.

- **BVT_MOSMS** *(Bluevia.Messagery.SMS.Client.BVT_MOSMS)*

MMS

Se muestra un esquema de estos, sus relaciones y sus métodos en la Figura 4-28.

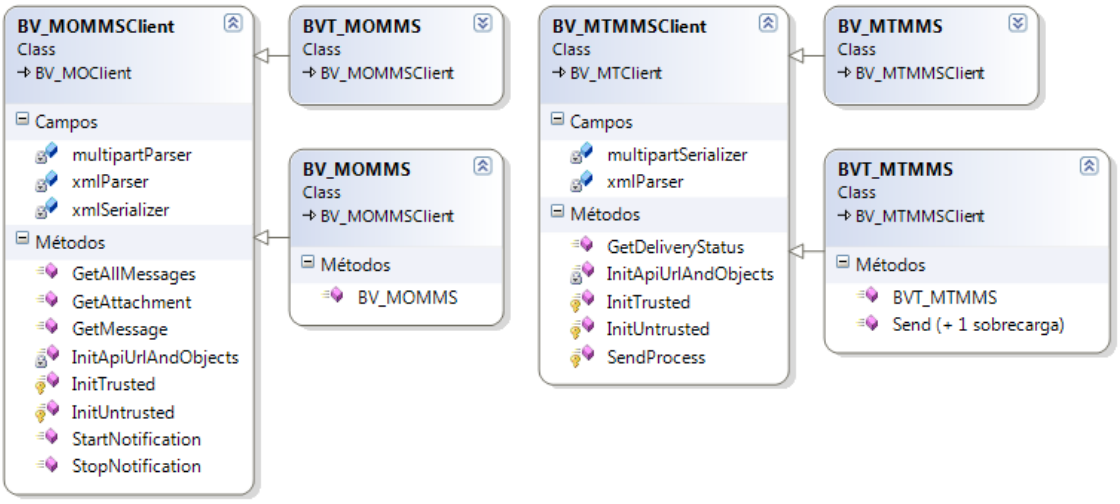


Figura 4-28 Diagrama de clientes de mensajería MMS

MOBILE TERMINATED

Estos clientes contienen a funcionalidad ligada a los procesos de envío de MMS a terminales móviles.

Salvo por el *endpoint* del servicio, y algunos parámetros que pueden enviarse con la operación *Send* –el MMS puede albergar contenido multimedia y un mensaje textual extra-; el proceso de las operaciones *Mobile Terminated* de MMS es igual al de SMS.

- **BV_MTMMSClient** *(Bluevia.Messagery.MMS.Client.BV_MTMMSClient)*

  Este es el cliente común de los servicios *Mobile Terminated* para mensajería MMS.

  Como en el caso del cliente común de SMS, algunos de los servicios necesitan "serializador" y "parseador" y otros no; por lo que se establecen los campos *multipartSerializer* y *xmlParser*, para poder conmutar entre ellos y *null.*

  - *SendProcess.* En este caso, el método no engloba la funcionalidad de varios servicios virtuales, sino que realiza el proceso de envío de un SMS para las versiones pública y privada del API.

    Será en los clientes finales, donde se sitúe el punto de entrada a este proceso.

    Usa la operación *BaseCreate*, y un "serializador" XML, para preparar la información del MMS a enviar.

    La particularidad de este método es la forma de recuperar la información de respuesta; ya que esta no viaja en el cuerpo del mensaje, sino en una de las cabeceras. Por ello, este método no usa "parseador" y directamente saca la información del objeto ***BV_Presponse*** que le lega el ***BaseClient***.

    De forma anecdótica cabe señalar, que a pesar de que la información básica de un MMS contiene un *subject,* a diferencia de los SMS que contienen un mensaje; el grupo de desarrolladores de los SDK decidió ofrecer a los usuarios la posibilidad de añadir un mensaje adicional, que sería recibido como *string* y convertido a archivo adjunto, en vez de obligarle a crear un objeto ***Attachment*** para ello.

  - *GetDeliveryStatus.* Este método recupera el estado de un mensaje previamente enviado.

    Usa la operación *BaseRetrieve,* a la URL que fue devuelta como respuesta al envío del MMS.

    Finalmente utiliza un "parseador" XML, para procesar la información del estado del envió.

- **BV_MTMMS** *(Bluevia.Messagery.MMS.Client.BV_MTMMS)*

  La entrada al *SendProcess* del ***BV_MMSMTClient*** se sitúa en este nivel; escribiéndose la funcionalidad relativa al acceso público en esta clase.

  - *Send.* Aquí se completan los datos de identificación de usuario, referentes al acceso público al servicio, y se invoca al *SendProcess* de ***BV_MMSMTClient*** para completar el proceso de envío de un MMS.

- **BVT_MTMMS** *(Bluevia.Messagery.MMS.Client.BVT_MTMMS)*

MOBILE ORIGINATED

Estos clientes contienen a funcionalidad ligada a los procesos de recepción de MMS originados por dispositivos móviles, y depositados en un buzón.

De forma similar a *Mobile Terminated,* los procesos de operaciones *Mobile Originated* de MMS, son muy parecidos a los de SMS; salvo porque al contener un MMS información multimedia que puede ser pesada, aparecen dos operaciones adicionales.

Como particularidad importante, al igual que en caso de SMS, hay que señalar que estos servicios no usan autorización de usuario en su versión pública, por los que los clientes no requerirán *Tokens.*

- **BV_MOMMSClient** *(Bluevia.Messagery.MMS.Client.BV_MOMMSClient)*

  - *GetAllMessages.* Este método descargará del buzón –definido por el parámetro *registrationId-* la información para recuperar de forma individual todos los MMS recibidos. Para ello usa la operación *BaseRetrieve* y procesa la respuesta con un "parseador" XML. Devolviendo finalmente un *array* de objetos ***MMSMessageInfo***.

    Opcionalmente se le puede pasar como parámetro un booleano que indique que además de la información para descargar los MMS, se desea también la información para descargar los archivos adjuntos que estos contengan, para descargarlos de forma individual.

  - *GetMessage.* Este método descargará del buzón el MMS indicado por el parámetro *messageId.* Utilizará la operación *BaseRetrieve*, y un "parseador" *multipart*, para procesar el MMS recuperado. Finalmente devolverá al usuario un objeto ***MMSMessage***.

  - *GetAttachment.* Este método descargará del buzón el archivo adjunto indicado de un MMS. Utilizará la operación *BaseRetrieve*, sin "parseador", y finalmente devolverá al usuario un objeto ***MIMEContent***, que habrá montado con el códifo de la Figura 4-29.

```
response.GetResponseHeaders().TryGetValue(HttpTools.ContetTypeKey, out
contentHeader);
string[] parts = contentHeader.Split(new Char[] { ';' });
return new MIMEContent(parts[1], parts[0], response.GetResponseBody());
```

Figura 4-29 Fragmento de código con el que se montan los objetos *MIMEContent* en el método *GetAttachment*

- *StartNotification.* Este método configura el buzón, para que reenvíe a un servidor –descrito por los parámetros que se envíen-, los MMS que le vayan llegando. Usa la operación *BaseCreate,* usando un "serializador" XML, para crear el cuerpo.

- *StopNotification.* Este método configura el buzón para parar el envío de MMS al servidor previamente definido. Usa la operación *BaseDelete*, por lo que no hay cuerpo que "serializar"; y tampoco necesita un "parseador" porque no hay respuesta –siempre que no haya error en el proceso-.

- **BV_MOMMS** *(Bluevia.Messagery.MMS.Client.BV_MOMMS)*

  Este cliente final, tan solo elimina de su construcción los parámetros *Tokens* de identificación de usuario.

- **BVT_MOMMS** *(Bluevia.Messagery.MMS.Client.BVT_MOMMS)*

Herramientas y clases auxiliares

SMS

Se muestra un esquema de estas y sus métodos en Figura 4-30.

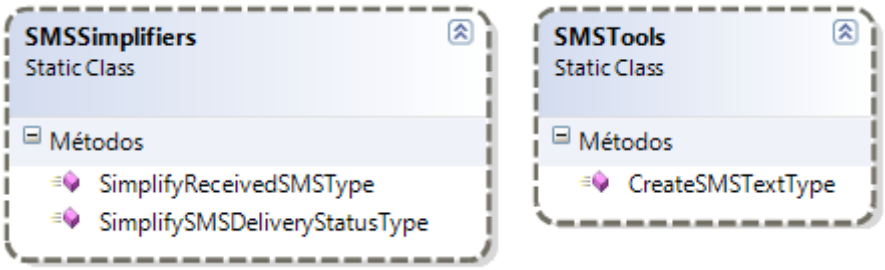


Figura 4-30 Diagrama de clases auxiliares de SMS

- **SMSSimplifiers** (Bluevia.Messagery.SMS.Tools.SMSSimplifiers)

  Leer comportamiento genérico de los simplificadores de objetos en: "Herramientas y clases auxiliares de API".

- **SMSTools** *(Bluevia.Messagery.SMS.Tools.SMSTools)*

    - *CreateSMSTextType*. Dado que para facilitar la labor al usuario, los datos del SMS a enviar, se reciben como parámetros simples; y como el formato que necesita *Bluevia* es un XML ligeramente complejo. Esta función monta con los datos recibidos un objeto de los definidos por la plataforma, para poder convertirlo a XML con un "parseador".

MMS

Se muestra un esquema de estas y sus métodos en Figura 4-31.

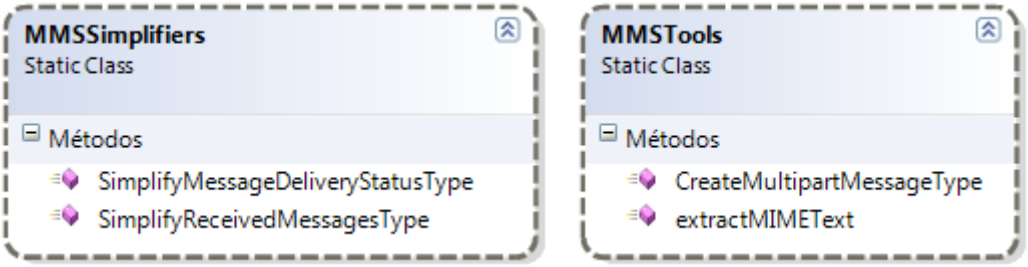


Figura 4-31 Diagrama de clases auxiliares de MMS

- **Tools.MMSSimplifiers** *(Bluevia.Messagery.MMS.Tools.MMSSimplifiers)*

    Leer comportamiento genérico de los simplificadores de objetos en la sección "Herramientas y clases auxiliares de API" del punto 4.2.2.3.

- **MMSTools** *(Bluevia.Messagery.MMS.Tools.MMSTools)*

    - *CreateMultipartMessageType*. Al igual que en el caso de los SMS, para facilitar la labor al usuario, los datos del mensaje a enviar, se reciben como parámetros simples; y como el formato que necesita *Bluevia* es un *multipart*. Esta función monta con los datos recibidos un objeto de los definidos por la plataforma, para poder convertirlo a *multipart* con un "parseador".

    - *extactMIMEText.* Este método se usa para transformar la selección enumerada del tipo de archivo a enviar, a un texto simple que pueda ser incrustado en el objeto multipart.

Objetos contenedores

Comunes

Se muestra un esquema de estos y sus campos en la Figura 4-32.

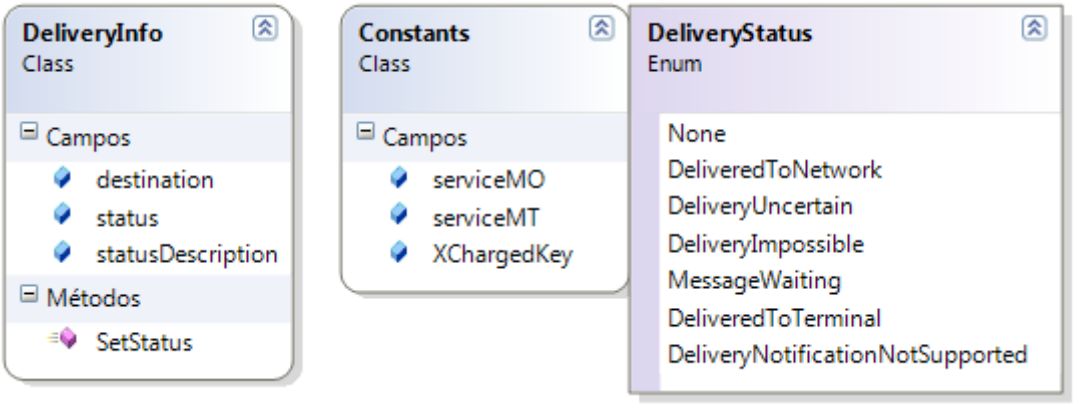


Figura 4-32 Diagrama de clases contenedoras y constantes de mensajería

- **DeliveryInfo** *(Bluevia.Messagery.DeliveryInfo)*

  Este objeto contiene los campos necesarios para definir el estado de envío de un SMS o de un MMS. Será usado como respuesta por los métodos *GetDeliveryStatus* de SMS y MMS MT.

  Contiene un método para generar el enumerado de estado, y la información textual de este, a partir de un string.

- **Constants** *(Bluevia.Messagery.Constants)*

  Además de las constantes comunes a SMS y MMS, define el enumerado:

  - **DeliveryStatus.** Este objeto enumera los posibles estados de envío para los mensajes en la red de *Bluevia.*

SMS

Se muestra un esquema de estos y sus campos en la Figura 4-33.

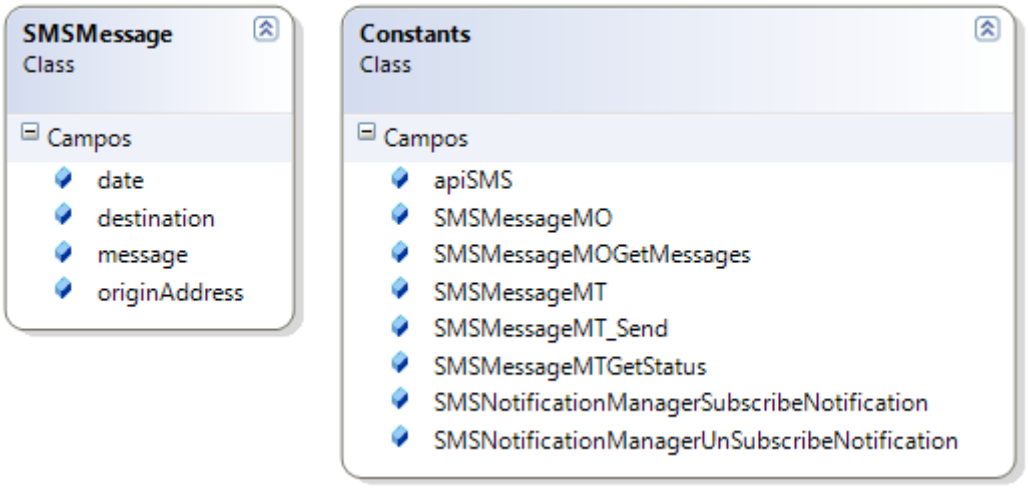


Figura 4-33 Diagrama de clases contenedoras y constantes de SMS

- **SMSMessage** *(Bluevia.Messagery.SMS.Schemas.SMSMessage)*

  Estos objetos contendrán la información de un SMS descargado de los buzones de *Bluevia.* Serán devueltos en *array* por la operación *GetAllMessages* de los clientes de SMS MO.

- **Schemas** *(Bluevia.Messagery.SMS.Schemas.Schemas)*

  Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

- **Constants** *(Bluevia.Messagery.SMS.Constants)*

  Leer la funcionalidad genérica de las clases de constantes en la sección "Objetos contenedores y constantes de API" del punto 4.2.2.3.

MMS

En este apartado cabe reseñar la cadena de transformaciones de objetos para enviar finalmente los archivos multimedia como adjuntos en un MMS:

1. El usuario crea un attachment (tipo y path) para enviar.
2. El cliente monta un objeto ***MultipartMessageType*** (que cambia el attachment por attachmentInfo, para pasar a texto el tipo de attachment), este objeto contiene:

- Los campos que se le ofrecen al usuario.
- La información de mensaje que puede llevar un SMS.
- Un texto extra.
- La lista de **attachment** virtuales.

Este ***MultipartMessageType*** se descompone por el ***BV_MultipartSerializer*** en una lista de ***AttachmentObjects*** genéricos (contenido del adjunto, y "metainformación"), que se pasarán al ***MultipartSerializer*** genérico.

Dentro de cada clase se extenderá la descripción del proceso.

Se muestra un esquema de estos objetos y sus campos en la Figura 4-34.

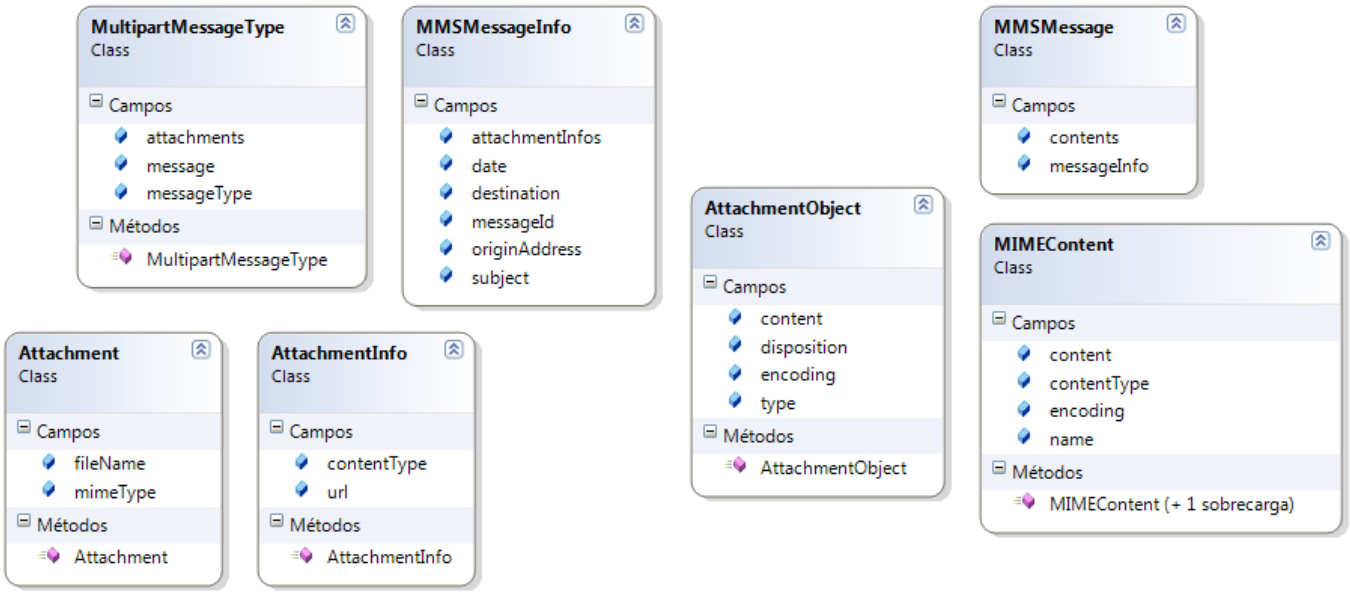


Figura 4-34 Diagrama de clases contenedoras de MMS

- **Attachment** *(Bluevia.Messagery.MMS.Schemas.Attachment)*

  Este objeto contiene la información de un archivo multimedia que quiera enviarse en un MMS. Deberá montarlo el usuario y pasarlo en *array* como uno de los parámetros del método *Send,* en los clientes de *MMS MT.*

  Contiene un *string* para definir la ruta del archivo, y un enumerado para definir uno de los tipos de archivo soportados por *Bluevia*.

- **MultipartMessageType** *(Bluevia.Messagery.MMS.Schemas.MultipartMessageType)*

  Dado que a la hora de enviar un MMS, al usuario se le pide que aporte la información en parámetros sencillos, este objeto encapsula toda esa información, de forma que pueda ser procesada fácilmente por un ***BV_MultipartSerializer***.

- **AttachmentInfo** *(Bluevia.Messagery.MMS.Schemas.AttachmentInfo)*

  Este objeto es equivalente al anteriormente descrito: ***Attachment***, con una diferencia; en tipo de archivo es descrito como *string en vez de como enumerado.*

  Se usa en dos situaciones:

  - En MT, durante el montaje del objeto que se serializará para el envío de MMS: ***MultipartMessageType***. Se convierten los objetos ***Attachment*** recibidos del usuario, en esta clase de objetos, para que puedan ser fácilmente serializados.
  - En MO, como parte de la respuesta del método *GetAllMessages:* ***MMSMessageInfo***; describiendo la información de los archivos adjuntos disponibles en el buzón. Y por extensión, en el simplificador que genera estas respuestas.

- **MMSMessageInfo** *(Bluevia.Messagery.MMS.Schemas.MMSMessageInfo)*

  Este objeto será la respuesta del método *GetAllMessages* de los clientes de MO, y contendrá la información que describe los MMS existentes en el buzón.

- **AttachmentObject** *(Bluevia.Messagery.MMS.Schemas.AttachmentObject)*

  Dado que la "serialización" de *multiparts* se realiza en dos fases, este objeto hace de interfaz entre ellas.

  - El ***BV_MultipartSerializer*** creará un objeto de esta clase, con cada parte del ***MultipartMessageType*** –creando en el proceso, los arrays de bytes de los archivos multimedia, a partir de sus rutas-.
  - Por otro lado el ***MultipartSerializer***, esperará una lista de estos objetos, para montarlos directamente en el archivo *multipart*.

- **MMSMessgae** *(Bluevia.Messagery.MMS.Schemas.MMSMessgae)*

  Esta clase es el MMS completo que genera el "parseador", para devolver directamente al usuario como respuesta del método *GetMessage* en los clientes MO.

- **MIMEContent** *(Bluevia.Messagery.MMS.Schemas.MIMEContent)*

  Esta clase contiene los campos del archivo adjunto completo, que es devuelto al usuario como respuesta del método *GetAttachment*; o en un array con otros adjuntos, dentro de un ***MMSMessage*** en la operación *GetMessage.*

- **Schemas** *(Bluevia.Messagery.MMS.Schemas.Schemas)*

  Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

Se muestra un esquema de estos objetos y sus campos en la Figura 4-35.

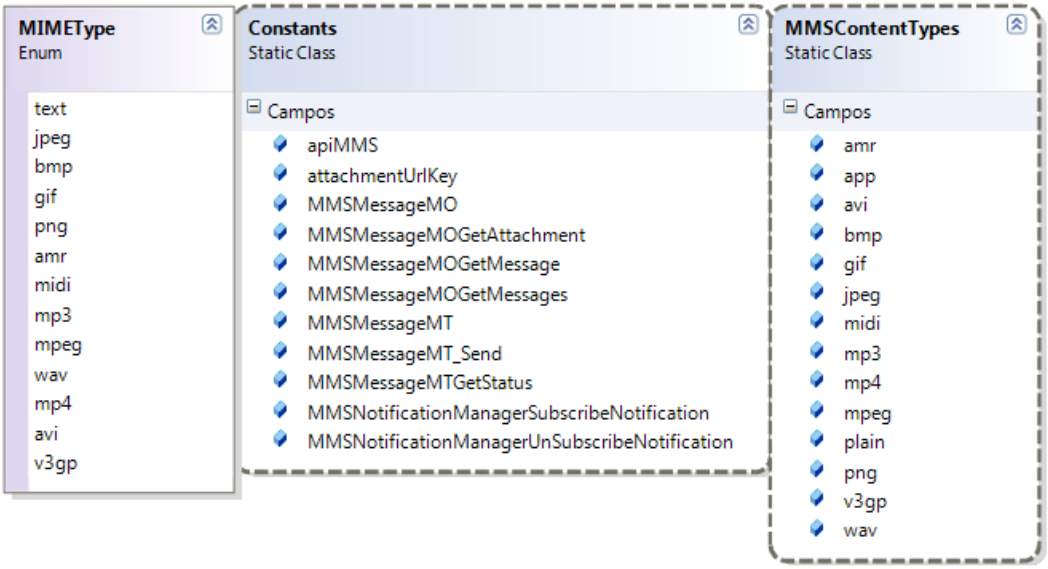


Figura 4-35 Diagrama de constantes y enumerados de MMS

- **Constants** *(Bluevia.Messagery.MMS.Constants)*

  Además de las constantes, define el enumerado y la clase:

  - **MIMEType**

    Es un enumerado que lista las opciones de archivos multimedia posibles en los MMS de *Bluevia,* para facilitarle al usuario la elección de estos.

  - **MMSContentTypes**

    Es una clase estática con las opciones de archivos multimedia posibles en los MMS de *Bluevia*, para definir el tipo/extensión de estas.

f. Payment

Este paquete contiene los clientes y objetos necesarios para invocar los servicios de pagos de *Bluevia.*

Además de lo descrito en la implementación de las clases de inteligencia de un API genérico, el paquete de *Payment* particulariza en lo siguiente:

Como en este API no se autoriza a la aplicación, si no a cada una de las transacciones; y como los servicios privados difieren bastante de los públicos; en vez de seguir la estructura general de un cliente común y dos clientes finales, el cliente final público de *Payment* extenderá del cliente común de *Oauth* -*BV_OauthClient*- para poder realizar autorizaciones de forma autónoma; mientras que el cliente final privado extenderá directamente del cliente base de *Bluevia* -*BV_BaseClient*-; como pudo verse en el punto *3.4.1.1* "Herencia en los módulos de inteligencia de *Bluevia*" .

Clientes de Payment

Se muestra un esquema de estos, sus relaciones y sus métodos en la Figura 4-36.

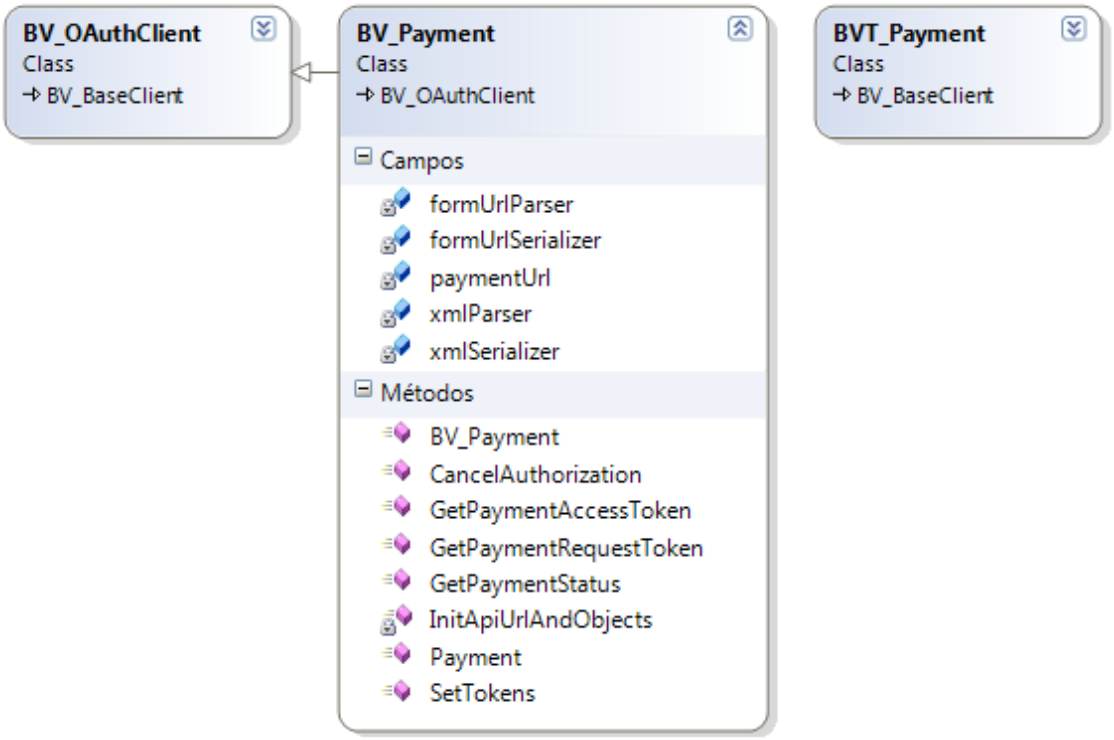


Figura 4-36 **Diagramas de clientes de *Payment***

- **BV_Payment** (Bluevia.Payment.Client.BV_Payment)

  Este cliente final, hereda la funcionalidad de recuperar *Tokens* del cliente común de *Oauth*, para autorizar las transacciones económicas que permite el API; y desarrolla el resto de la funcionalidad necesaria para realizar dichas transacciones.

  Por otro lado, los servicios propios de *Payment* se realizan por RPC a un *endpoint* distinto del de los demás API; por lo que el proceso de inicialización de URL también varía con respecto a los demás.

  Contiene dos "serializadores" y dos "parseadores" –*FormURL* y XML-, para poder alternarlos en los distintos métodos:

- *GetPaymentRequestToken.* Este método es la tercera entrada virtual –al *GetRequestTokenProcess* del ***BV_OauthClient***- para recuperar los *Tokens* temporales. En esta ocasión para una transacción.

  En él se seleccionan "parseador" y "serializador" *FormURL*, para que el código de *GetRequestTokenProcess* funcione correctamente; y se crea un diccionario de pares "Campo de *Payment*/Valor", para indicar la transacción que se quiere autorizar.

- *GetPaymentAccessToken.* Se trata de una entrada virtual al método *GetAccesToken* de *Oauth* que no especializa su comportamiento. Simplemente se ofrece para evitar posibles confusiones.

- *SetTokens.* Este método se ofrece para poder establecer los *Tokens* para invocar las operaciones *GetPaymentStatus* o *CancelAuthorization* sobre una transacción anterior –también Payment si el pago no se había realizado tras autorizarlo-.

- *Payment.* Este método RPC invoca el servicio de pago. Utiliza "serializador" y "parseador" XML, y la operación *BaseCreate.*

  Los datos de la transacción deben de ser los mismos que se utilizaron para autorizarla.

  Como particularidad sobre los servicios generales, este método fuerza el *timestamp* de la cabecera de HTTP "Authorization:Oauth", para que sea el mismo que el que contiene el objeto RPC del pago.

- *GetPaymentStatus.* Este método RPC recupera el estado de una transacción previa. Utiliza "serializador" y "parseador" XML, y la operación *BaseCreate.*

- *CancelAuthorization.* Este método RPC cancela la validez de la autorización de una transacción, es decir: los *Tokens*. Utiliza "serializador" y "parseador" XML, y la operación *BaseCreate.*

- **BVT_Payment** (Bluevia.Payment.Client.BVT_Payment)

Herramientas y clases auxiliares

Se muestra un esquema de estas y sus métodos en la Figura 4-37.

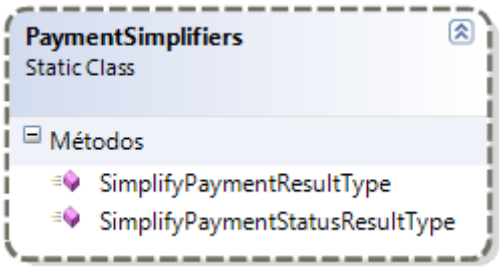


Figura 4-37 Diagrama del simplificador de *Payment*

- **PaymentSimplifiers** (Bluevia.Payment.Tools.PaymentSimplifiers)

  Leer comportamiento genérico de los simplificadores de objetos en la sección "Herramientas y clases auxiliares de API"del punto 4.2.2.3.

Objetos contenedores

Se muestra un esquema de estos objetos y sus campos en la Figura 4-38.

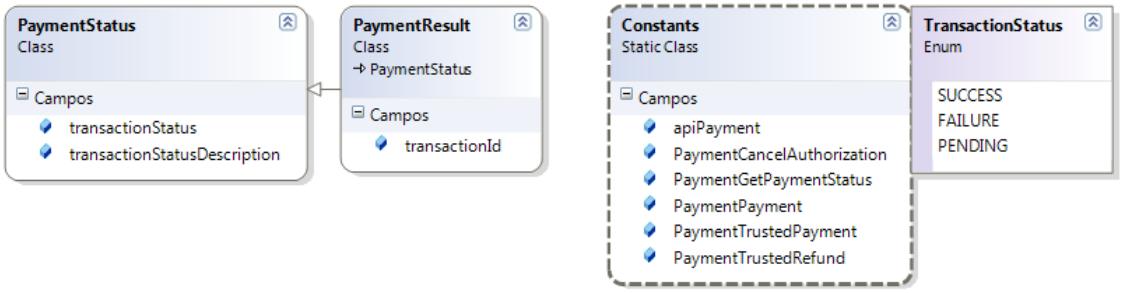


Figura 4-38 Diagrama de los objetos contenedores y constantes de *Payment*

- **PaymentStatus** *(Bluevia.Payment.Schemas.PaymentStatus)*

  Este objeto devuelto por el método *GetPaymentStatus* contiene los campos que describen el estado de una transacción; un enumerado ***TransactionStatus*** con el estado, y un *string* con la descripción de este.

- **PaymentResult** *(Bluevia.Payment.Schemas.PaymentResult)*

  Este objeto Devuelto por el método *Payment*, extiende al objeto ***PaymentStatus***, y amplía su información con un campo *transactionId,* que hace referencia a la transacción realizada, y permite preguntar por el estado de esta más adelante.

- **Schemas** *(Bluevia.Payment.Schemas.Schemas)*

  Leer la funcionalidad genérica de las clases ***Schemas*** en la seccion "Objetos contenedores y constantes de API" del punto 4.2.2.3.

- **Constants** *(Bluevia.Payment.Constants)*

  Además de las constantes, define el enumerado:

  - **TransactionStatus**. Este enumerado ofrece la selección de los estados posibles de una transacción. Se usa como campo en el objeto ***PaymentStatus***.

4.2.3 Unificación de la interfaz expuesta al usuario

Al finalizar la implementación de las librerías, se ha cuidado el aspecto de unificar la interfaz con la de los SDKs del resto de lenguajes, como se especificaba en uno de los requisitos exigidos; de modo que su manejo es equivalente al del resto, con los mismos clientes, métodos y parámetros.

En la Figura 4-39 puede observasre este hecho.

Instanciación de un cliente con la versión 1.6 del SDK de *PHP:*	**Instanciación de un cliente con la versión 1.6 del SDK de *.NET:***

$locationClient = new BV_Location BV_Mode::LIVE, "consumer_key", "consumer_secret", "access_token", "access_token_secret");
 	
Invocación de un método del cliente:
 	
$info = $locationClient->getLocation(100);
 	
Extracción de la información:
 	
$latitude = $info->coordinatesLatitude;
$longitude = $info->coordinatesLongitude;
	var locationClient = new BV_Location BVMode.LIVE, "consumer_key", "consumer_secret", "access_token", "access_token_secret");
	
	Invocación de un método del cliente:
	
	var info= locationClient.GetLocation(100);
	
	Extracción de la información:
	
	var latitude = info.coodinatesLatitude;
	var longitude = info.coodinatesLongitude;

Figura 4-39 Código comparativo de la creación de un cliente, la instanciación de uno de sus métodos, y la recuperación de la información obtenida; entre los SDKs de *PHP* y *C#*

Nota: En el código anterior, si se compara con la Figura 2-13 y la Figura 2-14, de: Limitaciones de las versiones anteriores del SDK para *.NET*, que comparaban el código para las versiones de 1.5; puede comprobarse como actualmente el manejo de los SDKs es prácticamente el mismo para todos los lenguajes: Instanciación de clientes, nombres de métodos y parámetros, etc. Y como se ha simplificado enormemente el manejo del SDK de *.NET*.

4.2.4 Problemas encontrados

A la hora de desarrollar las librerías, surgieron varias situaciones que supusieron cierta dificultad para poder ser resueltas. Se presentan a continuación en el orden en el que fueron apareciendo:

- El primer problema se plantea en la parte más baja de la estructura, a intentar prescindir -a diferencia del resto de lenguajes- de unas librerías externas que gestionen la conectividad y autorización *OAuth*. Para solucionarlo, se hace uso del objeto de conexión ofrecido por el *framework:* ***HttpWebRequest,*** que permite una configuración prácticamente total de conexiones HTTP; en combinación con la construcción del objeto ***OAuthManager*** para proporcionar toda la funcionalidad *OAuth* requerida por *Bluevia*.

- Otra cuestión estructural se planteaba en el problema de trasladar el diseño de las interfaces de "**Serializador**" y "**Parseador**" genéricas, que pudiesen ocuparse de preparar varios formatos diferentes en sus implementaciones específicas. Para ello los lenguajes de tipos dinámicos ***Ruby*** y ***PHP***, podían recibir parámetros de forma genérica, y procesarlos según fuese necesario; por lo que se decide acercar a dicha forma usando la funcionalidad de *Reflection* que ofrece ***C#***. De modo que pueden declarase parámetros sin un tipo específico en la interfaz, y definirlo más tarde en las implementaciones.

  Nota: El *framework 4* permite a *C#* la posibilidad de usar tipos dinámicos, pero como en el caso de los lenguajes no tipados, a costa de perder la ayuda en la depuración; por lo que decidió usarse la funcionalidad de Reflection en su lugar.

- Finalmente, y dado que los servicios de mensajería no pudieron ser probados en modo *LIVE* -funcionamiento real- hasta los últimos días del desarrollo, durante las primeras partes de este proceso, se empaquetaban los cuerpos de los mensajes en objetos textuales *String* por comodidad en la depuración. Estos *Strings* en ***C#*** poseen un formato adjunto de carácter, lo que provocaba que se enviasen *bytes* incorrectos a la hora de empaquetar objetos multimedia. Más tarde, cuando se comenzaron a probar los servicios de recuperación de MMS, y no podían construirse correctamente algunos archivos, se cambió el proceso de empaquetado, guardando los archivos directamente en *arrays* de *bytes;* y los textos al mismo formato también, usando la codificación por defecto del sistema para evitar cambios en los caracteres.

4.3 IMPLEMENTACIÓN DE LAS DEMOS

El paquete del SDK debe de ofrecer el código de aplicaciones demostrativas funcionales, de al menos un servicio por API, ya que para un desarrollador es más intuitivo aprender a usar rápidamente una librería con ejemplos prácticos de esta.

Para ello, se modifica y amplía el paquete de ejemplos que ya se ofrecía en la versión 1.5 del SDK, para funcionar con las nuevas librerías; aprovechando para mejorar la presentación de la información, y la documentación contenida en el código.

Este paquete de ejemplos es una versión simplificada -y externa a la solución del código de las librerías-, y con datos no sensibles precargados, para invocar solo servicios *Sandbox* de la aplicación de test que se ha ido desarrollando desde la versión 1.2 del SDK, para verificar el desarrollo y funcionamiento de las librerías.

En su momento, se elige este sistema de demos en consola en detrimento de otras opciones como la de demos en sistema web; ya que la consola aporta inmediatez de ejecución, un código más directo y con menos interferencias.

4.3.1 Descripción de clases de los ejemplos públicos

4.3.1.1 Clase principal e interfaz de pruebas.

Puede verse un esquema de estas en la Figura 4-40.

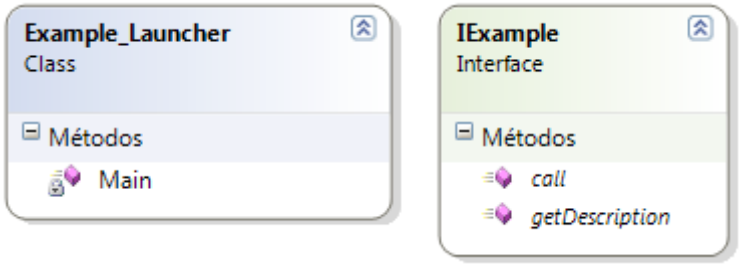


Figura 4-40 Diagrama de clases principales del paquete de ejemplos

- **Example_Launcher** *(BlueviaExamples.Example_Launcher)*

  Esta es la clase principal del paquete de ejemplos. Contiene lógica sencilla para cargar, presentar y ejecutar las demos; en un menú de consola como interfaz.

  En la imagen de la Figura 4-41 puede apreciarse el aspecto del menú.

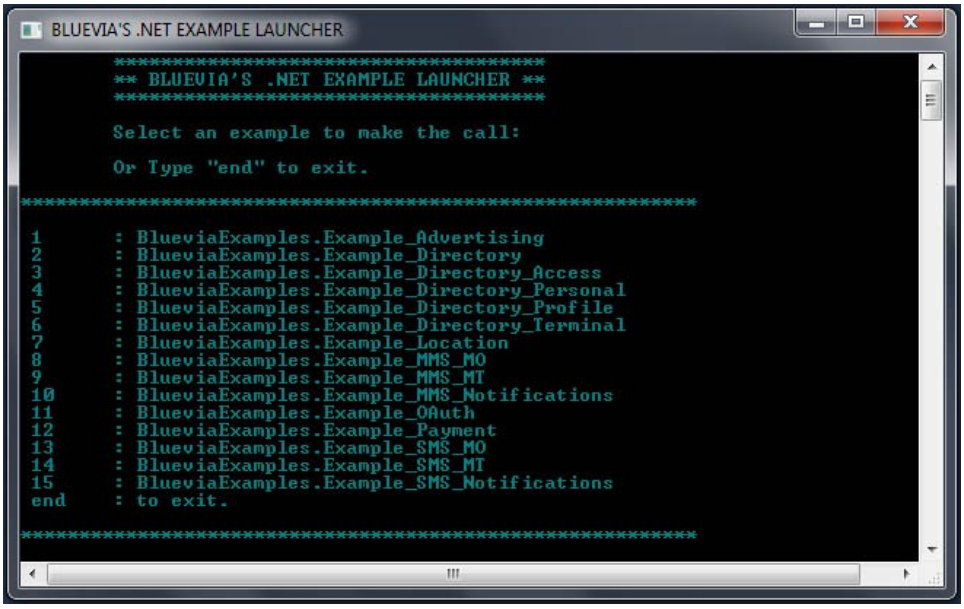


Figura 4-41 Vista del menú principal de la aplicación de demos Example_Launcher

- Main.

  Al iniciar, esta clase configura los parámetros visuales de la consola, y carga una instancia por cada uno de los ejemplos existentes.

  Tras ello dibuja el menú, y espera a recibir por entrada estándar la selección correcta. En ese momento recupera a través del método *getDescription* la información del ejemplo seleccionado, y presenta un nuevo menú en el que explica su funcionamiento.

  Por lo general, todos los ejemplos pueden lanzarse sin modificaciones en modo *Sandbox*, gracias a que el paquete contiene *Consumer Keys* y *Tokens* precargados, de la aplicación de una cuenta para pruebas. Pero en algunos casos los servicios necesitan información delicada de la cuenta -que no puede ser compartida con los usuarios-, o información relativa al equipo donde se ejecutan los ejemplos; por lo que se pedirá al usuario que complete estos datos extra para poder realizar la prueba.

  El flujo del lanzador de ejemplos esperará a que el usuario termine de leer las instrucciones de cada prueba -e introducir la información en caso necesario-, y ejecute la prueba. Cuando recibe la señal, invoca el método *call* del ejemplo, que realiza las peticiones descritas en su descripción, y presenta los datos recuperados.

  Cuando la prueba termina, el menú vuelve al estado inicial presentando una vez más la lista de ejemplos.

  En la Figura 4-42 se puede ver la salida por consola de la ejecución de un ejemplo:

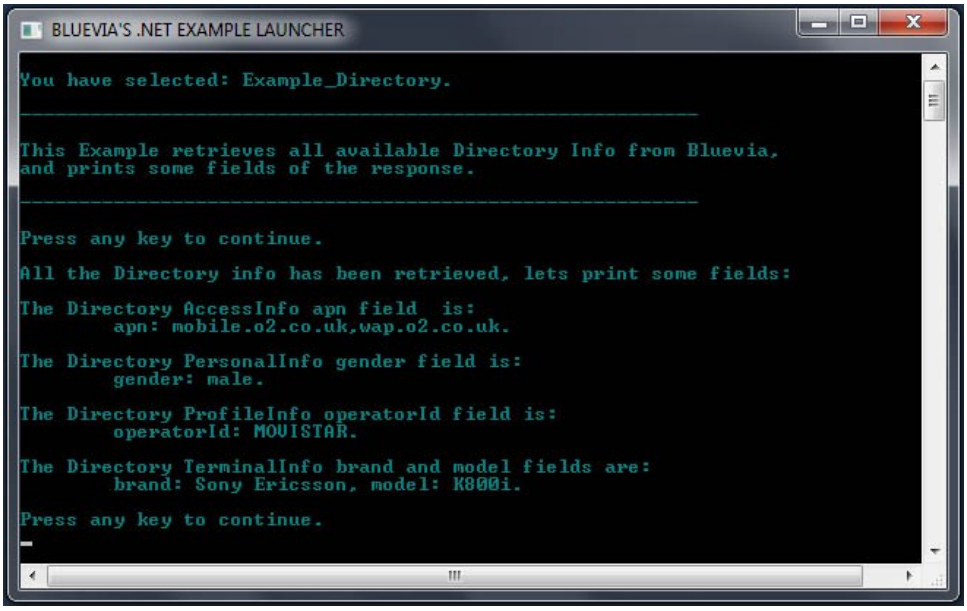


Figura 4-42 **Vista de la pantalla de ejecución del ejemplo de *Directory***

- **IExample** *(BlueviaExamples.IExample)*

  Esta interfaz define la funcionalidad que debe de contener un ejemplo del paquete de demos.

  - *getDescription.* Este método define la devolución de la descripción del ejemplo, para que pueda ser presentada al usuario por el ***Example_Launcher***.
  - *Call.* Este método define la invocación de los servicios que abarque el ejemplo. Recibirá por parámetros las credenciales para poder ejecutarlos.

4.3.1.2 Los ejemplos: Example_API_Tipo

Estas clases contienen las descripciones y el código para ejecutar uno o varios servicios del API relativo. Implementan para ello los métodos de la interfaz ***IExample***:

- *getDescription.* Aquí simplemente se guardará un *string* con la información del ejemplo, que será devuelto al ***Example_Launcher*** cuando el método sea invocado.
- *Call.* Este método invoca los servicios presentados en la descripción. Contiene comentarios detallados de todo el proceso de invocación para hacer de estas clases buenas herramientas didácticas. Se crea un cliente del API relativo al ámbito del ejemplo, y se invocan los servicios presentando todos los parámetros -comentando su obligatoriedad-.

  Finalmente se devuelven los resultados.

Las siguientes clases siguen la funcionalidad descrita en ***Example_API_Tipo***; en las descripciones se enunciarán los servicios invocados.

Nota: Pueden verse los fragmentos relevantes del código de cada prueba, y el resultado de la invocación en el punto: 4.4, VALIDACIÓN DE LAS LIBRERÍAS**.**

- **Example_Advertising** *(BlueviaExamples.Example_Advertising)*

  Esta clase realiza la invocación para recuperar un anuncio de imagen relacionado con la palabra "Bluevia". Además emplea *protectionPolicy* y adPresentation -a pesar de no ser obligatorios-, para ilustrar al usuario el manejo de los parámetros de política de madurez y tipo de anuncio.

  Tras recuperar la información, imprime la URL de la imagen del *banner* publicitario, y la URL a la que redirigir al pulsar sobre el *banner*.

- **Example_Directory** *(BlueviaExamples.Example_Directory)*

  Esta clase realiza una petición para recuperar todos los sets de información del API de directorio.

  Tras recuperar la respuesta, imprime un campo de cada set de información.

- **Example_Directory_Access** *(BlueviaExamples.Example_ Directory_Access)*

  Esta clase realiza una petición para recuperar todos los campos del set de información de acceso del API de directorio.

  Tras recuperar la respuesta imprime tres campos de información.

- **Example_Directory_Personal** *(BlueviaExamples.Example_ Directory_Personal)*

  Esta clase realiza una petición para recuperar todos los campos del set de información personal del API de directorio.

  Tras recuperar la respuesta imprime un campo de información.

- **Example_Directory_Profile** *(BlueviaExamples.Example_ Directory_Profile)*

  Esta clase realiza una petición para recuperar solo dos campos del set de información del contrato, del API de directorio. Así, en el código se crean los parámetros necesarios para filtrar los campos de la respuesta -identificador de operador, y lenguaje-, para ilustrar al usuario sobre su uso en el API de *Directory*.

  Tras recuperar la respuesta imprime los campos recuperados.

- **Example_Directory_Terminal** *(BlueviaExamples.Example_ Directory_TermianI)*

  Esta clase realiza una petición para recuperar solo dos campos del set de información del terminal, del API de directorio. Así, en el código se crean los parámetros necesarios para filtrar los campos de la respuesta -marca y modelo del terminal-, para ilustrar al usuario sobre su uso en el API de *Directory*.

  Tras recuperar la respuesta imprime los campos recuperados.

- **Example_Location** *(BlueviaExamples.Example _Location)*

  Esta clase realiza una petición para recuperar la localización del terminal móvil, con una precisión de 500 metros, especificando dicha magnitud en la llamada –a pesar de ser un parámetro opcional- para ilustrar al usuario sobre el uso del cliente.

  Tras recuperar la respuesta, imprime la precisión obtenida, latitud y longitud.

- **Example_MMS_MO** *(BlueviaExamples.Example_ MMS_MO)*

  Esta clase realiza varias peticiones del API de MMS, relacionadas con la recuperación de mensajes.

  La prueba consiste en el envío y recuperación de una imagen vía MMS. Para ello, la descripción pide al usuario que escriba en el código fuente una ruta válida a una imagen en su disco duro.

  Una vez lanzada la prueba, se crearán dos clientes, un ***BV_MTMMS*** para realizar el envío del mensaje a un buzón, y un ***BV_MOMMS*** para recuperar el mensaje.

  El mensaje se recuperará en dos operaciones, siguiendo el funcionamiento del API. La primera descargará la lista de mensajes existentes en el buzón para conseguir el identificador del mensaje, y la segunda descargará el mensaje propiamente dicho.

  La clase ofrece también código comentado para mostrar cómo sería la recuperación del archivo adjunto en solitario, para que sirva de ejemplo.

- **Example_MMS_MT** *(BlueviaExamples.Example_ MMS_MT)*

  Esta clase realiza el envío de un mensaje y después recupera su estado, usando el API de MMS.

  Durante la inicialización de la prueba, se le informará al usuario de la necesidad de escribir en el código fuente la ruta a un archivo de imagen para poder enviarlo.

- **Example_MMS_Notifications** *(BlueviaExamples.Example_ MMS_Notifications)*

  Esta clase realiza la subscripción, para recuperar en un servidor propio los MMS dirigidos al buzón. Tras ello, el ejemplo invoca el método de baja del servicio.

La descripción informa al usuario de que deben de ser provisionados datos válidos en el código de la prueba.

- **Example_OAuth** *(BlueviaExamples.Example _OAuth)*

  Esta clase contiene el código para realizar un proceso guiado de autorización de *OAuth*, recuperando unos *Tokens* que simbolicen el permiso de un usuario para que la aplicación realice peticiones a los servicios de *Bluevia* en su nombre.

  Para animar al desarrollador a desenvolverse en *Bluevia*, la descripción informa al usuario de que deberá usar sus propias credenciales en lugar de las de la aplicación de pruebas -por lo tanto tendrá que crearse una cuenta de desarrollador, y dar de alta al menos una aplicación-. Además le informa de que deberá proporcionar durante el proceso de la ejecución del ejemplo el verificador que simboliza el permiso de un usuario -que puede ser el mismo- para que la aplicación realice peticiones en su nombre.

  Así, esta prueba realiza una operación de petición de *Tokens* temporales, y manda al usuario a autorizar la aplicación a la página oportuna para conseguir el código de verificación.
  Tras ello, realiza la petición de *Tokens* definitivos.

- **Example_Payment** *(BlueviaExamples.Example _Payment)*

  Esta clase realiza el proceso completo de un pago -autorización, ejecución y recuperación de estado-, usando el API de *Payment*.

  Para ello, como la autorización de la operación necesita de un proceso completo de *OAuth*, la descripción informa al usuario de que debe de proporcionar datos propios para poder ejecutar de forma correcta la prueba.

  La prueba, realiza un proceso de *OAuth* para transacciones, para autorizar la operación.

  Tras ello, ejecuta el pago, recupera el estado de este; y finalmente cancela la autorización.

- **Example_SMS_MO** *(BlueviaExamples.Example_ SMS_MO)*

  Esta clase realiza varias peticiones del API de SMS, relacionadas con la recuperación de mensajes.

  La prueba consiste en el envío y recuperación de un SMS.

  Para ello, una vez lanzada la prueba se crearán dos clientes, un ***BV_MTSMS*** para realizar el envío del mensaje a un buzón, y un ***BV_MOSMS*** para recuperar el SMS.

- **Example_SMS_MT** *(BlueviaExamples.Example_ SMS_MT)*

  Esta clase realiza el envío de un mensaje y después recupera su estado, usando el API de SMS.

- **Example_SMS_Notifications** *(BlueviaExamples.Example_ SMS_Notifications)*

  Esta clase realiza la subscripción, para recuperar en un servidor propio los SMS dirigidos al buzón. Tras ello, la prueba invoca el método de "desubscripción".

  La descripción informa al usuario de que deben de ser provisionados datos válidos en el código del ejemplo.

4.3.2 Diferencias entre los ejemplos públicos y privados

Al igual que en el caso público, la librería *Trusted* de acceso privado contiene también ejemplos de uso de los servicios.

Estos ejemplos están diseñados y empaquetados de la misma forma que en el caso público, salvando la diferencia de las autorizaciones de *OAuth* -inexistentes en el acceso privado-, por lo que aparecen las siguientes diferencias:

- En el método *Main* de la clase ***Example_Launcher*** no existirán *Tokens*, y por lo tanto no serán recibidos por parámetro en las clases ***Example_API_Tipo***.
- No se realizará el proceso de autorización en ***Example_Payment***.
- No existirá una prueba de *OAuth* -***Example_OAuth*** desaparece en los ejemplos privados-.

4.4 VALIDACIÓN DE LAS LIBRERÍAS

Para validar las librerías, debe de comprobarse que se cubren todas las especificaciones descritas por *Bluevia* -ver resumen de las más importantes en 7.2- a la hora de invocar los servicios disponibles; en todos sus modos operativos, y con toda la variedad posible de opciones.

Para ello, antes de realizar la entrega de una versión, se ha utilizado la aplicación de test.

Esta aplicación se crea inicialmente -en la versión 1.2 del SDK- como una herramienta para realizar invocaciones rápidas usando las diferentes opciones de la librería; pero aprovechando el desarrollo desde cero de la nueva versión del SDK, se integra a la aplicación de test en la solución del proyecto, y se convierte en una herramienta de apoyo al desarrollo.

La aplicación consiste en una interfaz de línea de comandos simple que ofrece una lista de test contenidos en clases individuales, enfocadas a probar varios métodos relativos a un ámbito específico -por ejemplo probar el correcto funcionamiento de varios "parseadores" o invocar los servicios de un API usando diferentes datos-, de modo que con ella puede depurarse el funcionamiento del código de la librería, y verificar rápidamente la invocación correcta de los servicios.

Durante el desarrollo de la librería, la aplicación de test era lanzada contra un entorno de preproducción que contenía la versión en desarrollo de *Bluevia* 1.6, para validar las funcionalidades nuevas y realizar pruebas intensivas en el modo *LIVE*; aunque en ocasiones algunos servicios no estaban implementados y había que realizar las pruebas contra el entorno comercial ofrecido en aquel momento -producción- de *Bluevia 1.5*.

Una vez pasadas las pruebas con la herramienta de test, el SDK era sometido a pruebas de validación por parte de una Auditoría externa, lo que garantizaba el correcto funcionamiento de las librerías antes de ser finalmente publicadas.

Para poder ilustrar la validez de la invocación de servicios por parte de la librería, de forma resumida; se presentan un poco más adelante los fragmentos representativos del código que ejecuta las pruebas contenidas en la aplicación de demos -recordemos que la aplicación de demos es una versión simplificada de la aplicación de test, y que las invocaciones hacen referencia a todos los parámetros posibles a pesar de que no se usen-, junto con una captura de la prueba, seguida de los mensajes de red generados -ver proceso de captura en el anexo 7.4.1-.

Por otro lado, se recuerda que la aplicación de test considera también todos los servicios ofrecidos con varias combinaciones de parámetros; además de contener credenciales de cuentas *LIVE* y *TEST* para verificar el funcionamiento de las librerías en los máximos escenarios posibles.

Realiza también algunas combinaciones de "parámetros-servicios-modos" no permitidas por las especificaciones; para validar que la librería impide la invocación de los servicios en esos supuestos.

4.4.1 Validación de *Advertising*

- **A través de *Example_Advertising*:**

En este ejemplo se puede observar la invocación del servicio de recuperación de anuncios, usando el método de autenticación *3-legged*, y los parámetros opcionales de directiva de madurez, y palabras clave. El código de invocación del método y sus parámetros se ilustra en la Figura 4-43, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-44 y Figura 4-45.

```
adResponse = client.GetAdvertising3L(
                    adSpace: "BV15125", //MANDATORY
                    country: null, //Optional
                    adRequestId: null, //Optional
                    adPresentation: TypeId.image, //Optional
                    keywords: new string[] { "Bluevia" }, //Optional
                    protectionPolicy: ProtectionPolicy.low, //Optional
                    userAgent: "none" //Optional
                    );
```

Figura 4-43 Código de invocación de *Example_Advertising*

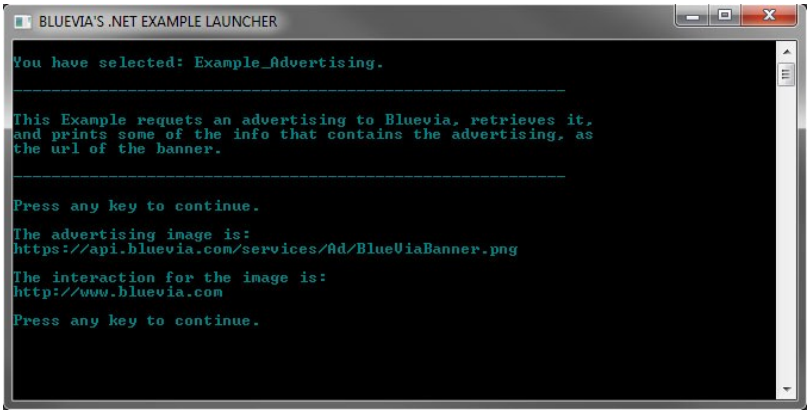


Figura 4-44 Captura de la consola al ejecutar *Example_Advertising*

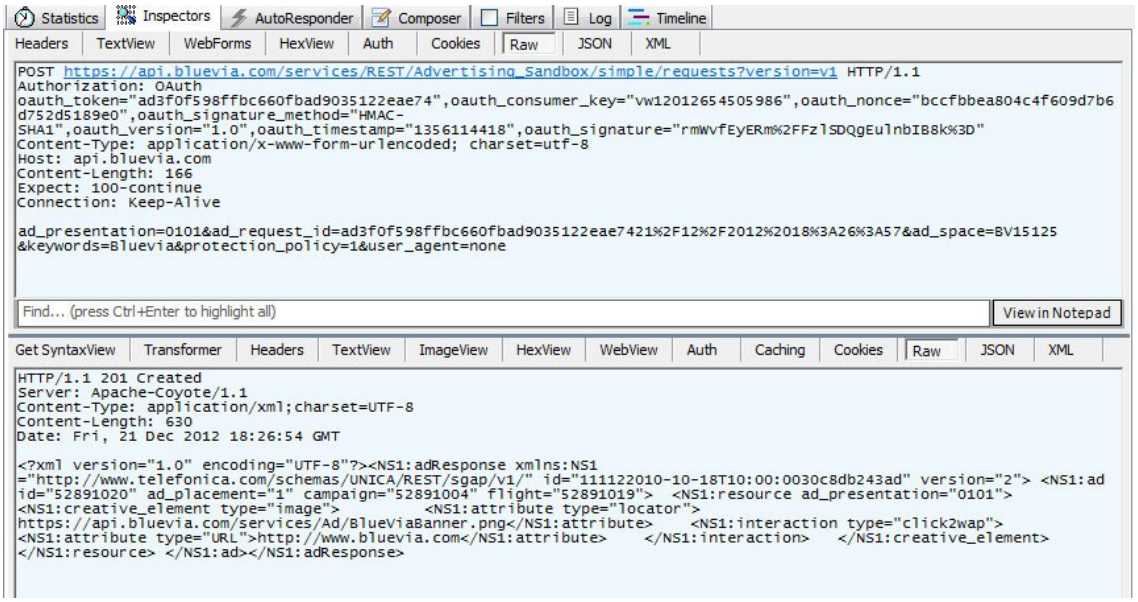


Figura 4-45 Captura de las trazas de red de la ejecución de *Example_Advertising*

4.4.2 Validación de *Directory*

- **A través de *Example_Directory:***

En este ejemplo se puede observar la invocación del servicio de datos de directorio, recuperando toda la información disponible. El código de invocación del método y sus parámetros se ilustra en la Figura 4-46, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-44 y Figura 4-48.

```
userInfo = client.GetUserInfo(
                dataSet: null //Optional
                );
```

Figura 4-46 Código de invocación de *Example_Directory*

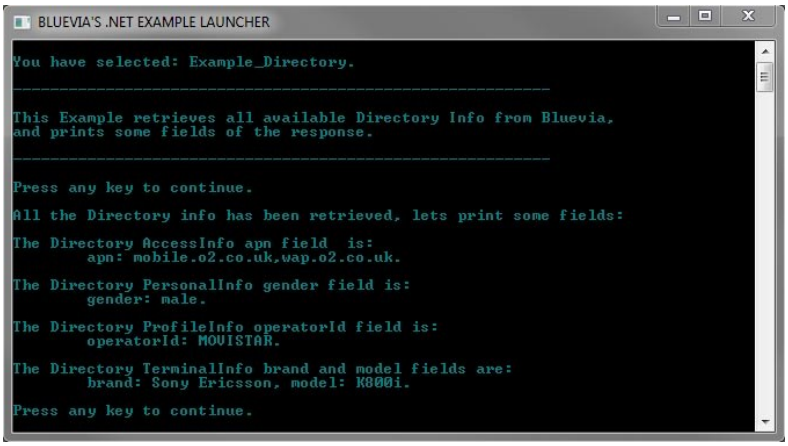


Figura 4-47 Captura de la consola al ejecutar *Example_Directory*

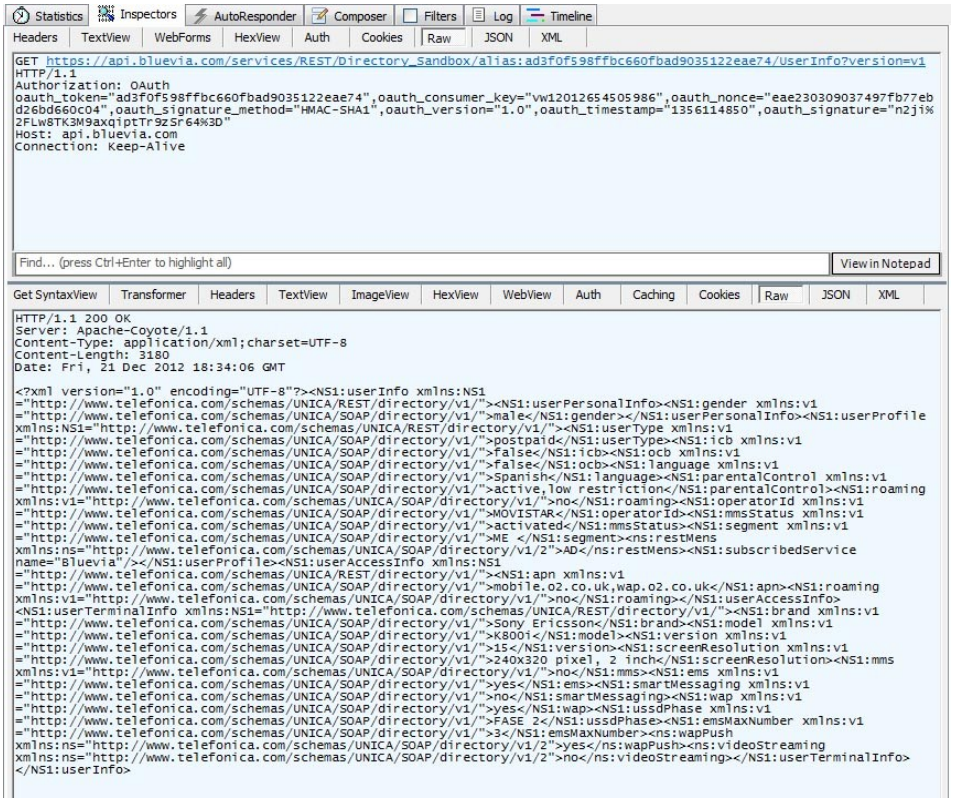


Figura 4-48 Captura de las trazas de red al ejecutar *Example_Directory*

- **A través de *Example_Directory_Access:***

En este ejemplo se puede observar la invocación del servicio de datos de directorio, recuperando la información relativa al acceso. El código de invocación del método y sus parámetros se ilustra en la Figura 4-49, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-50 y Figura 4-51.

```
accessInfo = client.GetAccessInfo(
                fields: null //Optional
                );
```

Figura 4-49 Código de invocación en *Example_Directory_Access*

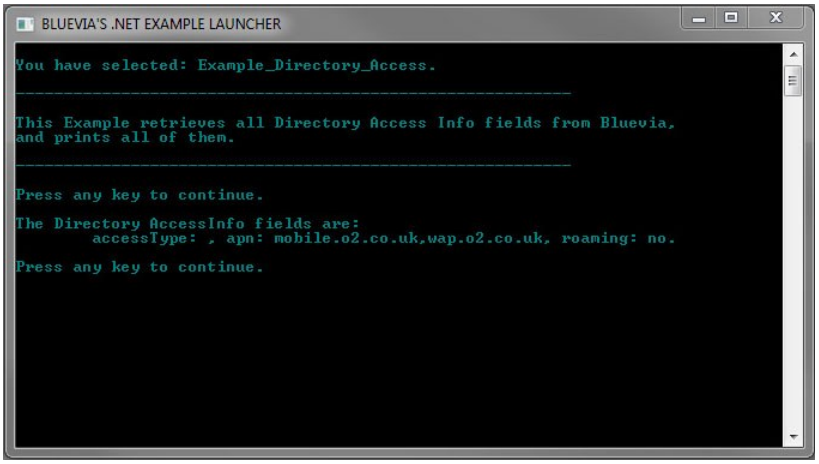


Figura 4-50 Captura de la consola al ejecutar *Example_Directory_Access*

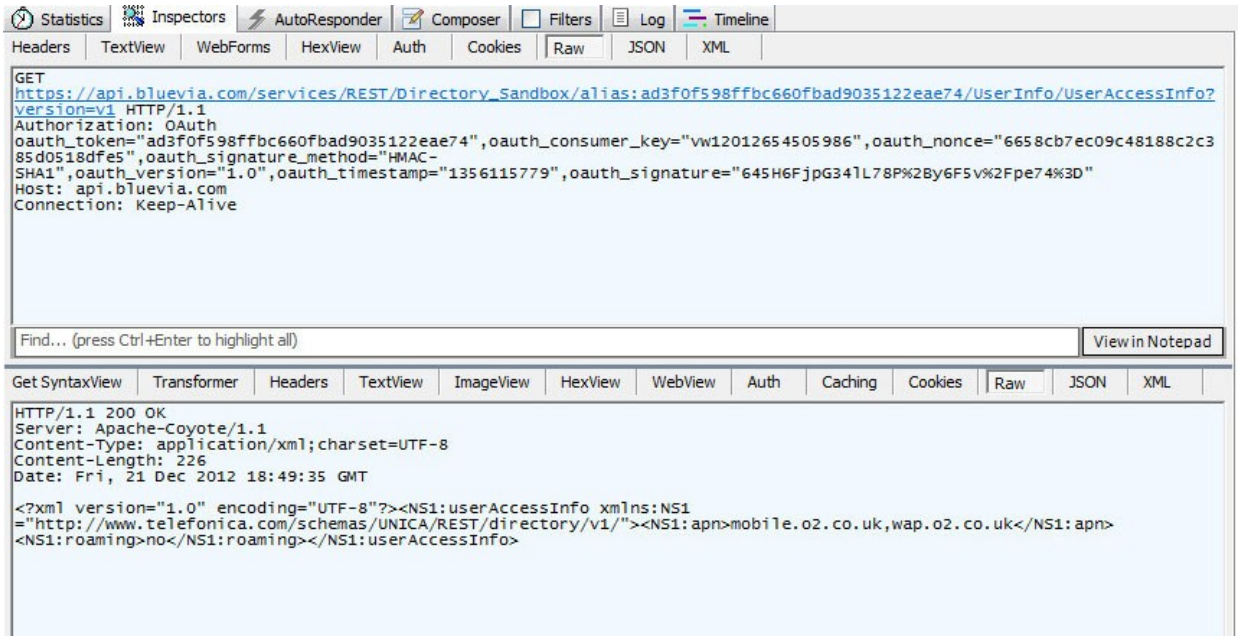


Figura 4-51 Captura de las trazas de red al ejecutar *Example_Directory_Access*

- **A través de *Example_Directory_Personal:***

En este ejemplo se puede observar la invocación del servicio de datos de directorio, recuperando la información personal del usuario de la línea. El código de invocación del método y sus parámetros se ilustra en la Figura 4-52, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-53 y Figura 4-54.

```
personalInfo = client.GetPersonalInfo(
                fields: null //Optional
                );
```

Figura 4-52 Código de invocación en *Example_Directory_Personal*

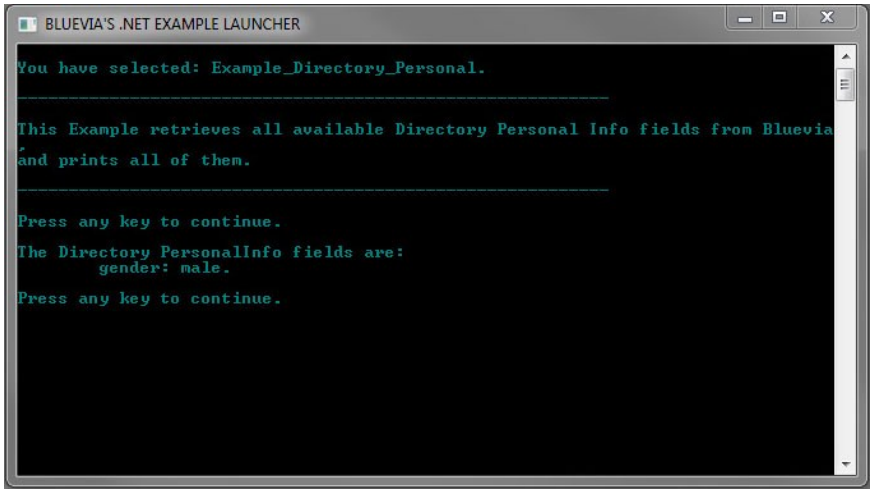


Figura 4-53 Captura de la consola al ejecutar *Example_Directory_Personal*

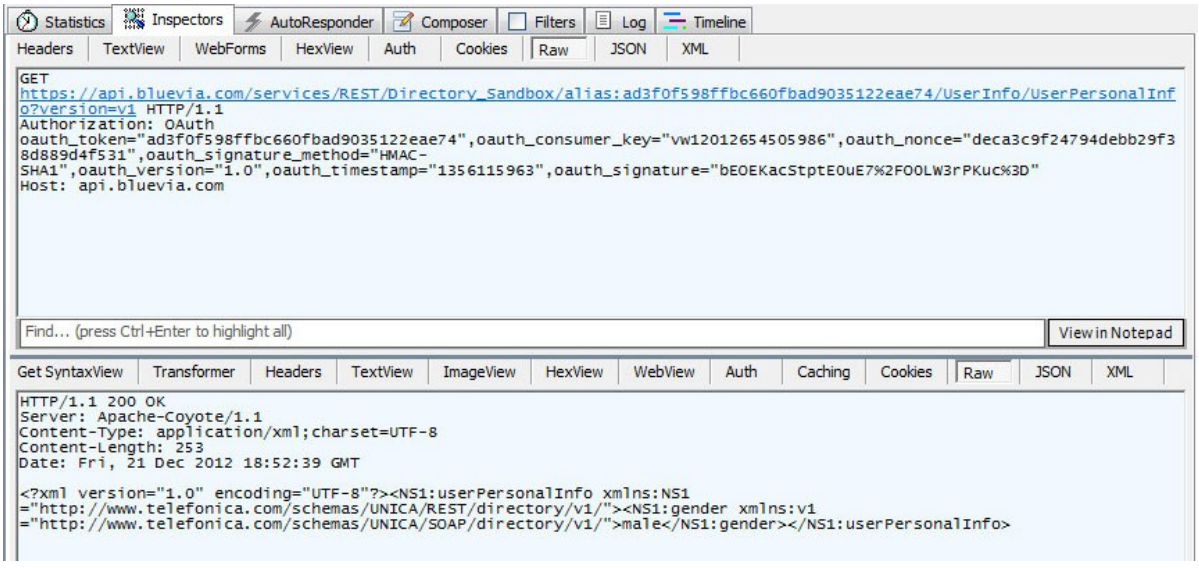


Figura 4-54 Captura de las trazas de red al ejecutar *Example_Directory_Personal*

- **A través de *Example_Directory_Profile:***

En este ejemplo se puede observar la invocación del servicio de datos de directorio, recuperando parte de la información relativa al perfil del contrato de la línea. El código de invocación del método y sus parámetros se ilustra en la Figura 4-55, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-56 y Figura 4-57.

```
profileInfo = client.GetProfileInfo(
        fields: new ProfileFields[]{ProfileFields.operatorId,ProfileFields.language}//Optional
        );
```

Figura 4-55 Código de invocación en *Example_Directory_Profile*

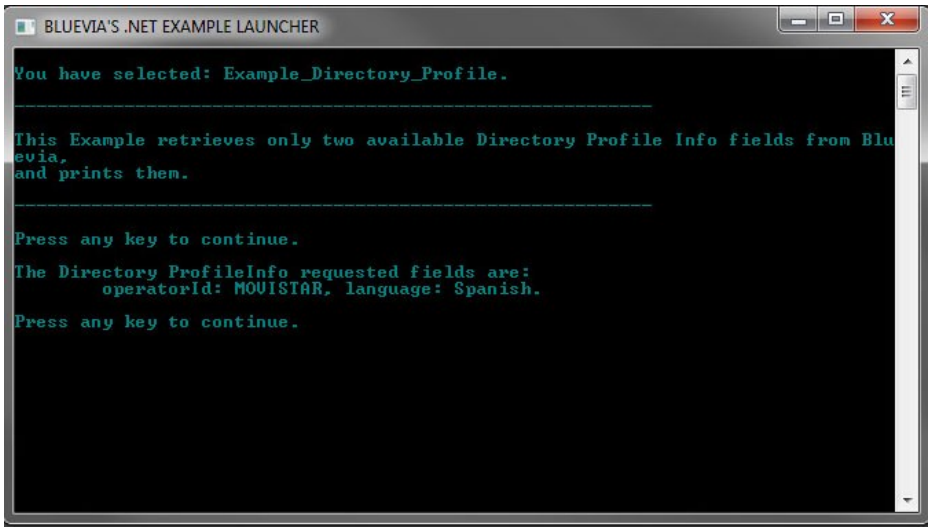


Figura 4-56 Captura de la consola al ejecutar *Example_Directory_Profile*

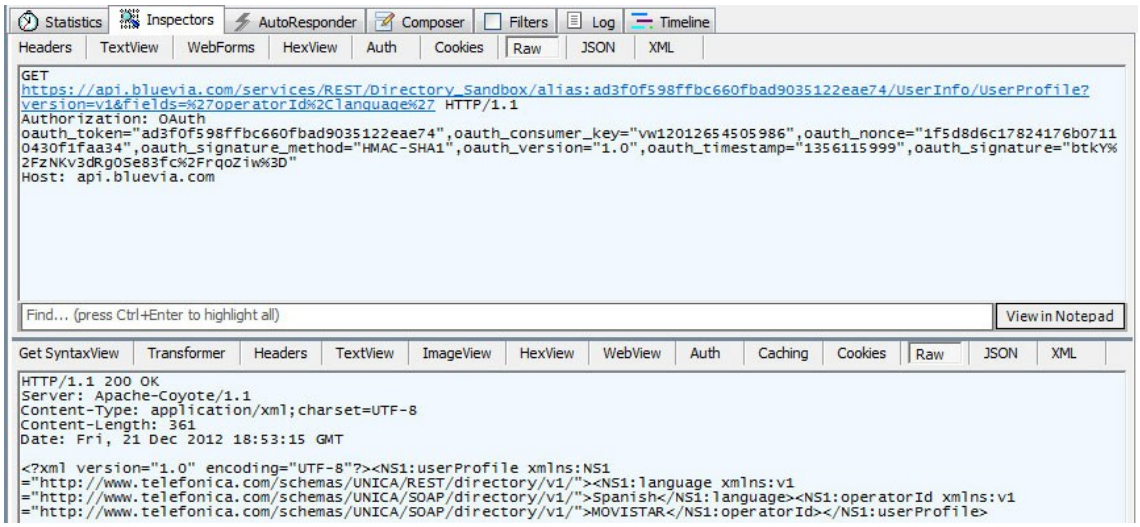


Figura 4-57 Captura de las trazas de red al ejecutar *Example_Directory_Profile*

- **A través de *Example_Directory_Terminal:***

En este ejemplo se puede observar la invocación del servicio de datos de directorio, recuperando parte de la información relativa al terminal asociado a la línea. El código de invocación del método y sus parámetros se ilustra en la Figura 4-58, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-59 y Figura 4-60.

```
terminalInfo = client.GetTerminalInfo(
        fields: new TerminalFields[]{TerminalFields.brand,TerminalFields.model} //Optional
        );
```

Figura 4-58 Código de invocación en *Example_Directory_Terminal*

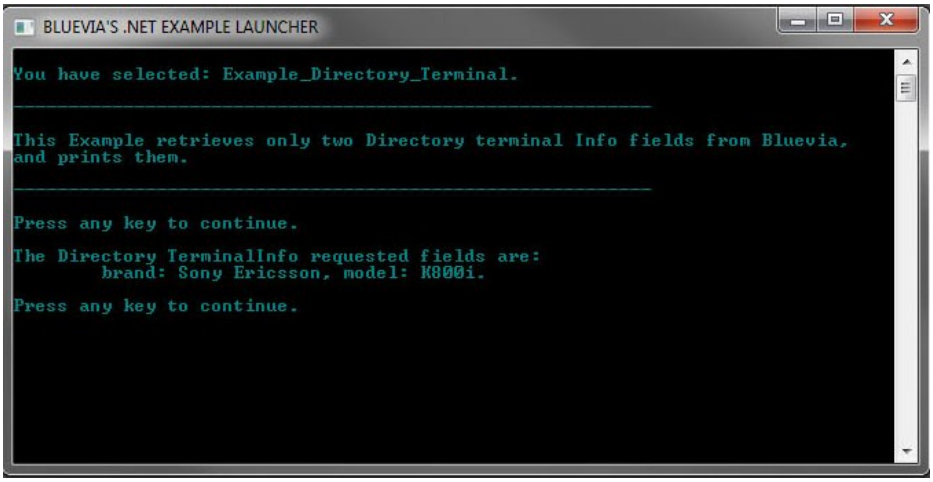


Figura 4-59 Captura de la consola al ejecutar *Example_Directory_Terminal*

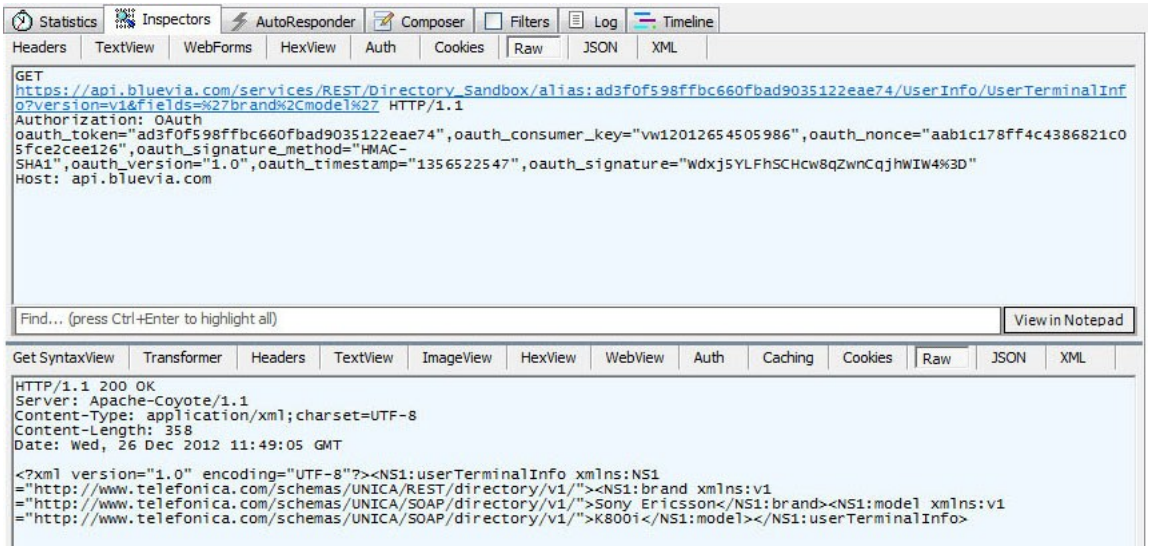


Figura 4-60 Captura de las trazas de red al ejecutar *Example_Directory_Terminal*

4.4.3 Validación de *Location*

- **A través de *Example_Location:***

En este ejemplo se puede observar la invocación del servicio de localización por red telefónica, restringiendo la precisión mínima a 500 metros.El código de invocación del método y sus parámetros se ilustra en la Figura 4-61, mientras que la captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras:Figura 4-62 y Figura 4-63.

```
locationInfo = client.GetLocation(
                accuracy:500 //Optional
                );
```

Figura 4-61 Código de invocación en *Example_Location*

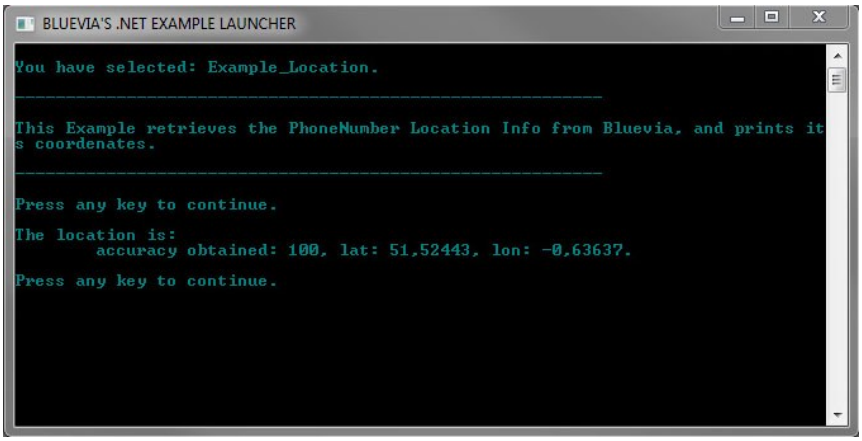


Figura 4-62 Captura de la consola al ejecutar *Example_Location*

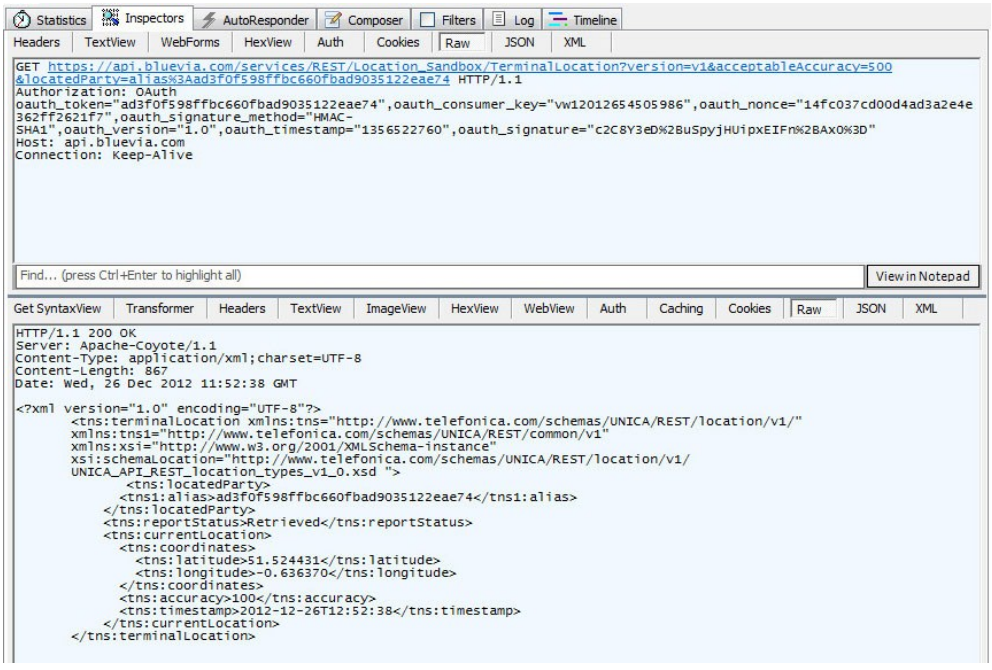


Figura 4-63 Captura de las trazas de red al ejecutar *Example_Location*

4.4.4 Validación de MMS

- **A través de *Example_MMS_MT:***

En este ejemplo se puede observar el envío de un MMS con dos archivos adjuntos -un texto y una imagen-, y la recuperación de su estado. La validación se realiza entonces en dos partes:

La primera con el envío del mensaje como puede verse en el código de la Figura 4-64.

```
string statusId = client.Send(
        destination: "54666112233", //MANDATORY
        subject: "SANDBLUEDEMOS This is a Dummie MMS Subject for MMS_MO", //MANDATORY
        message: "Optional text attachment", //Optional
        attachments: new Attachment[] //Optional
{
//TYPE THE PATH TO THE jpeg FILE
new Attachment("D:\\pic.jpg", MIMEType.jpeg)
},
        endpoint: null, //Optional
        correlator: null //Optional
        );
```

Figura 4-64 Código del envío de un MMS con un cliente MMS_MT, de *Example_MMS_MT*

Y la segunda con la recuperación del estado del envío, como muestrael código de la Figura 4-65.

```
deliveryInfos = client.GetDeliveryStatus(
        messageId: statusId //MANDATORY
);
```

Figura 4-65 Código de recuperación del estado de envío del MMS, de *Example_MMS_MT*

La captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-66, Figura 4-67 y Figura 4-68.

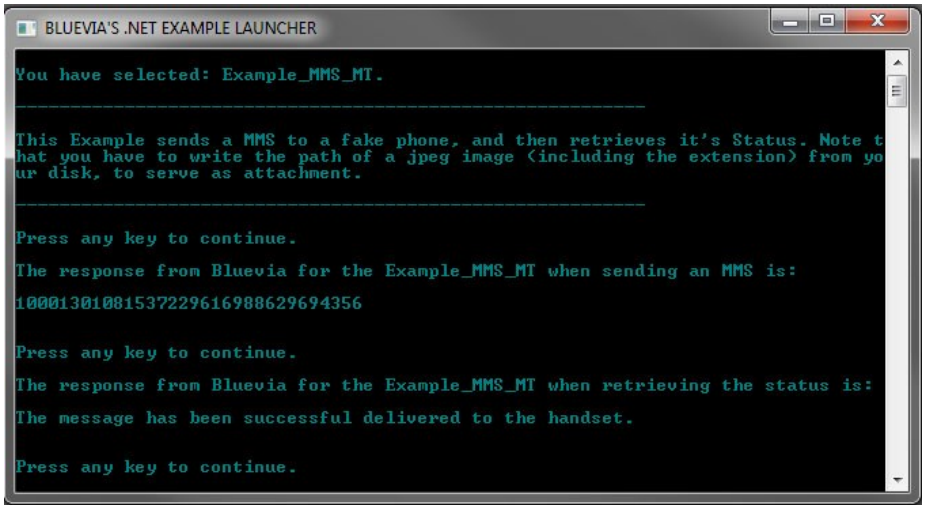


Figura 4-66 Captura de la consola al ejecutar *Example_MMS_MT*

Los mensajes de red relativos al envío del MMS se muestran a continuación. Puede observarse el formato MIME del mensaje, y los datos binarios de la imagen.

Después aparecen los mensajes relativos a la invocación del servicio para la recuperación del estado.

Figura 4-67 Captura de las trazas de red al enviar un MMS

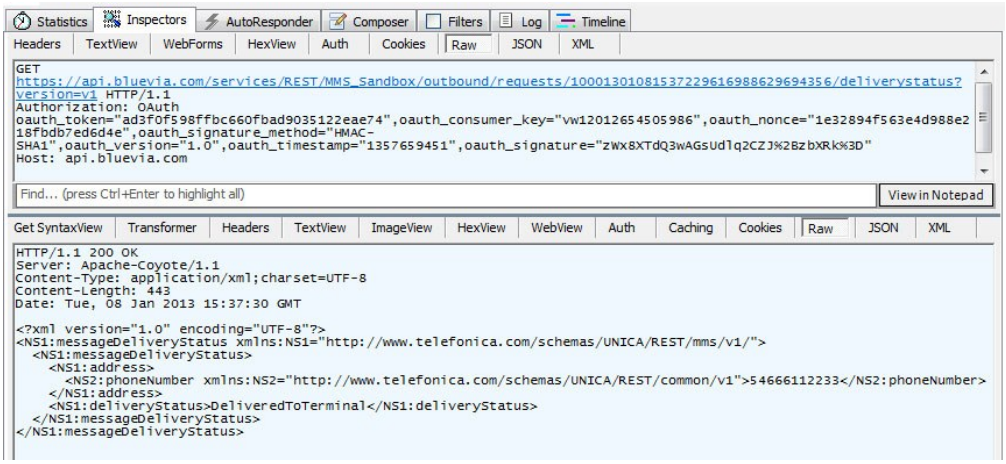


Figura 4-68 Captura de las trazas de red al recuperar el estado del envío de un MMS

En este caso, además de comprobar la invocación de los servicios, se ha considerado la codificación correcta de los caracteres en el envío de archivos multimedia para evitar problemas en los posibles cambios de plataformas.

Para ello se realiza la prueba en modo *LIVE* para enviar el mensaje a un dispositivo móvil, resultando exitosa al visualizarse correctamente los archivos en el dispositivo.

- **A través de *Example_MMS_MO:***

Esta prueba envía un MMS al número corto de un buzón, luego descarga la lista de mensajes del buzón, y finalmente descarga el primer mensaje existente.

El primero de los pasos, salvo en el valor del parámetro *destination* -que en este caso hace referencia al número corto de un buzón de pruebas-, es idéntico al envío de MMS mostrado en **MMS MT**, por lo que se expondrá el código de invocación, pero no la captura del mensaje de red. Los dos pasos siguientes si se muestran en su totalidad.

En la Figura 4-69 puede observarse que el código de envío del mensaje solo varía en el número de *destination*:

```
string statusId = client.Send(
        destination: "546780", //MANDATORY
        subject: "SANDBLUEDEMOS This is a Dummie MMS Subject for MMS_MO", //MANDATORY
        message: "Optional text attachment", //Optional
        attachments: new Attachment[] //Optional
                    {
                        //TYPE THE PATH TO THE jpeg FILE
                        new Attachment("D:\\pic.jpg", MIMEType.jpeg)
                    },
        endpoint: null, //Optional
        correlator: null //Optional
);
```

Figura 4-69 Código de envío de un MMS a un buzón, de *Example_MMS_MO*

Una vez enviado el mensaje con el cliente de **MMS MT**, se recupera la lista de mensajes del buzón con el cliente de **MMS MO**, dado que en el modo *Sandbox* no es posible la descarga

individual de archivos adjuntos, esta prueba ignora dicha posibilidad poniendo el parámetro *attachUrl* a falso, como se muestra en el código de la Figura 4-70.

```
inboxMessages = clientMO.GetAllMessages(
                    registrationId: "546780", //MANDATORY
                    attachUrl: false //Optional
                    );
```

Figura 4-70 Código de recuperación de la lista de MMS del buzón, de *Example_MMS_MO*

Con la lista nuestro poder, se selecciona el identificador del primer mensaje, y se descarga completo. La Figura 4-71 contiene el códiiigo que realiza dicha acción.

```
messageIdentifier = inboxMessages[0].messageId;
…
message = clientMO.GetMessage(
                    registrationId: "546780", //MANDATORY
                    messageId: messageIdentifier //MANDATORY
                    );
```

Figura 4-71 Código de recuperación del primer MMS del buzón, de *Example_MMS_MO*

La captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-72, Figura 4-73 y Figura 4-74.

Puede observarse que el MMS descargado. Es el mismo mensaje enviado en la prueba de **MMS MT** en formato MIME, pero al haber sido procesado en la recepción por los servidores de la red de Telefónica, los *boundaries* cambian. En cualquier caso, el contenido binario de los adjuntos es el mismo.

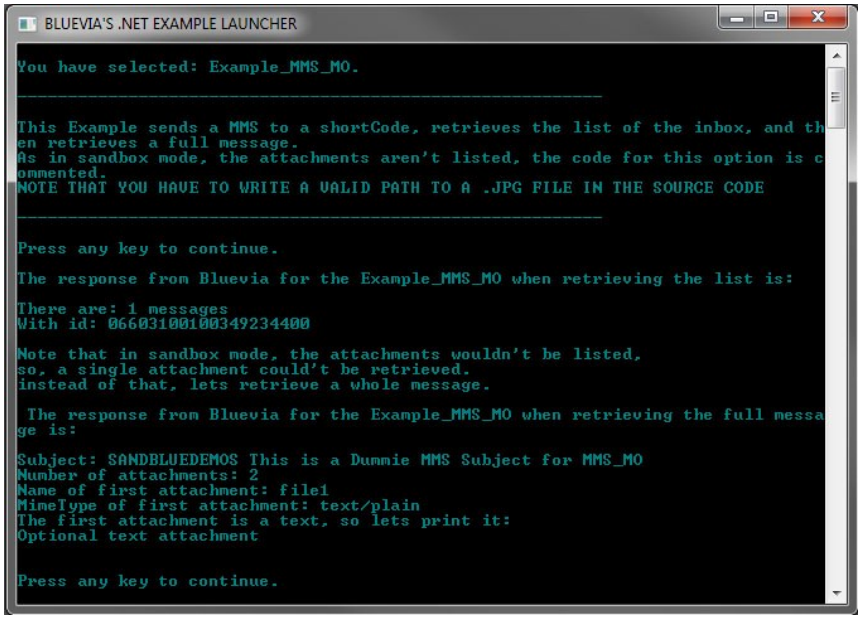


Figura 4-72 Captura de la consola al ejecutar *Example_MMS_MO*

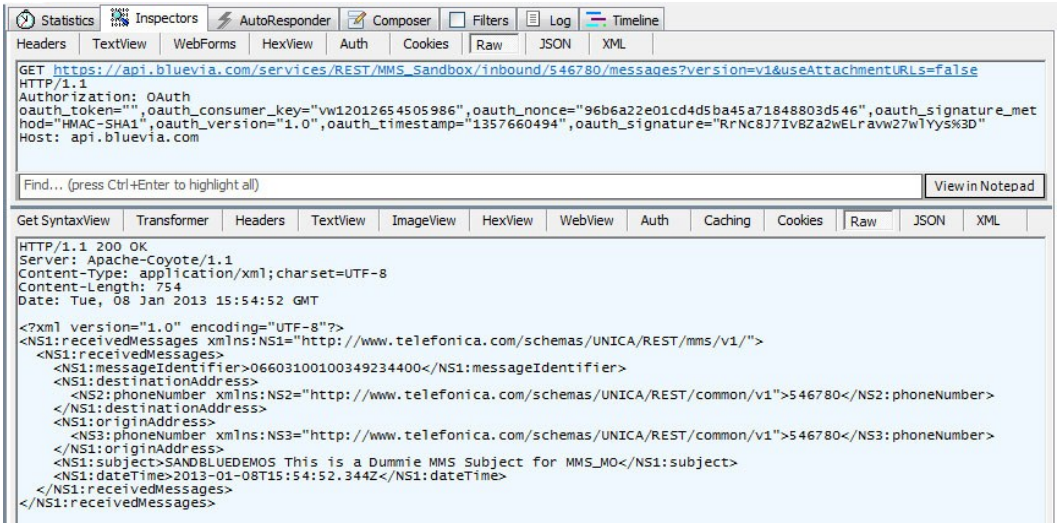


Figura 4-73 Captura de las trazas de red al recuperar la lista de mensajes existentes en el buzón

Figura 4-74 Captura de las trazas de red de la descarga de un mensaje completo, desde un buzón de MMS

Además de realizar pruebas para verificar el correcto funcionamiento de la invocación de los distintos servicios, en **MMS MO** también se ha considerado la recuperación correcta de los archivos multimedia adjuntos. Este requisito se ha validado creando correctamente el archivo recuperado en el propio disco para confirmar que la codificación de caracteres es correcta.

Se comprobaron dos posibilidades según el origen de los archivos depositados en el buzón: archivos enviados desde un dispositivo móvil, y archivos enviados desde la librería (como continuación de la validación de **MMS MT**). Resultando exitosas al visualizarse correctamente los archivos en el equipo.

- **A través de *Example_MMS_Notifications:***

En la prueba se realiza una subscripción y una baja para el servicio de notificaciones en la recepción de mensajes MMS. Como el código necesita modificación por parte del usuario que pretenda ejecutarlo, se aporta información relativa a la aplicación de pruebas *MessageryApisSharpPFCApp* -*Consumer Key:* aK12092580770877 -, para validar el funcionamiento. El código de invocación del método y sus parámetros se ilustra en la Figura 4-75.

```
string subscribe = client.StartNotification(
                    phoneNumber: "546780", //MANDATORY
                    endpoint: "https://MessageryApisSharpPFCApp/trial", //MANDATORY
                    criteria: "SANDApisSharpPFCAppKey", //MANDATORY
                    correlator : "correlator" //MANDATORY
                    );
```

Figura 4-75 Código de *Example_Notifications* modificado para dar de alta el servicio de notificaciones de MMS

El código de invocación del método de baja y sus parámetros se ilustra en la Figura 4-76.

```
bool unsusbscribe = client.StopNotification(
                   correlator: "correlator" //MANDATORY
                   );
```

Figura 4-76 Código de *Example_Notifications* modificado para dar de baja el servicio de notificaciones de MMS

La captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-77, Figura 4-78 y Figura 4-79.

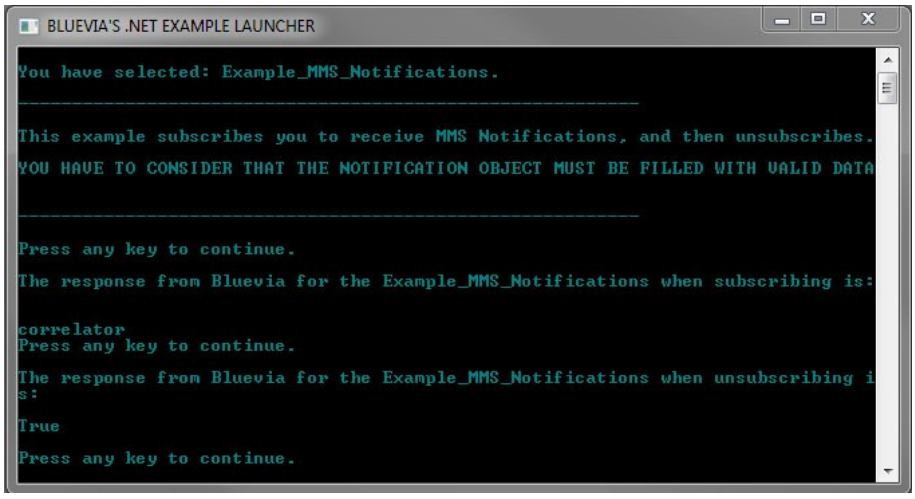


Figura 4-77 Captura de la consola al ejecutar *Example_MMS_Notifications*

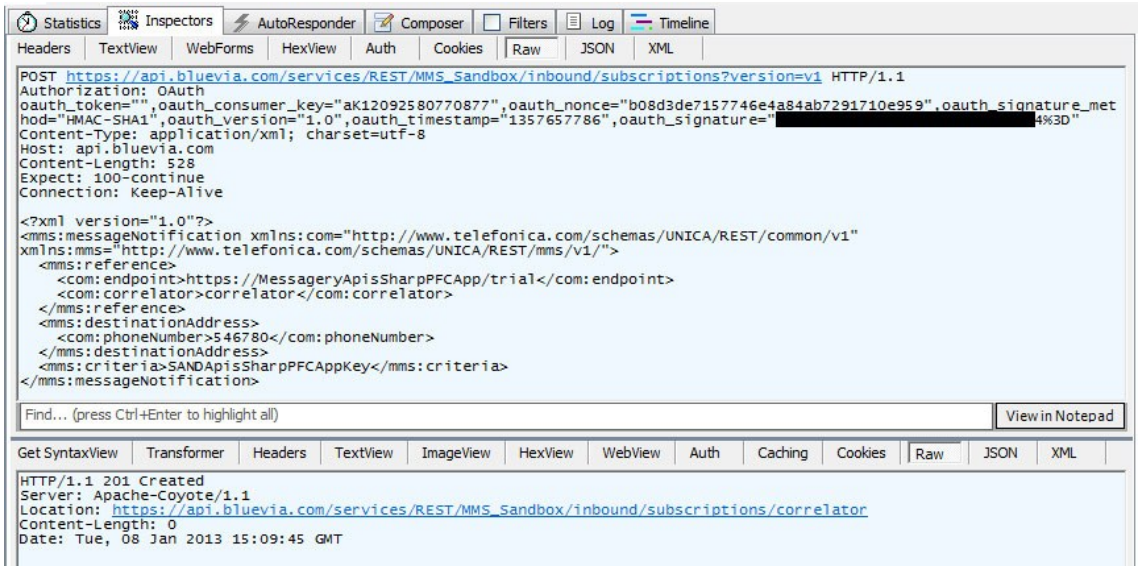


Figura 4-78 **Captura de las trazas de red de la subscripción a las notificaciones de MMS**

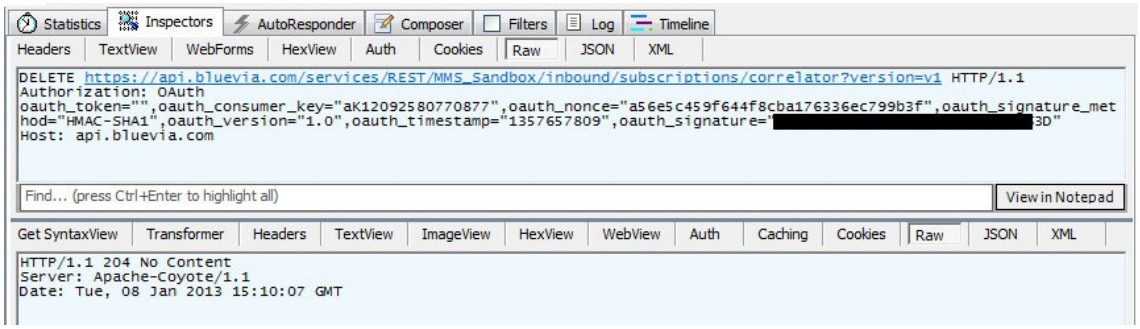


Figura 4-79 **Captura de las trazas de red de la baja para las notificaciones de MMS**

4.4.5 Validación de *OAuth*

- **A través de *Example_OAuth:***

En el ejemplo se realiza el proceso de *OAuth* para recuperar unos *Tokens* que simbolicen el permiso de un usuario para que la aplicación realice peticiones a los servicios de *Bluevia* en su nombre. El código de la prueba se modifica para ilustrar este caso, utilizando la aplicación de prueba: *FreeApisSharpPFCApp -Consumer Key:* Rc12060644955223-, y realizando la autorización en modo *LIVE*. El código de invocación del método y sus parámetros se ilustra en la Figura 4-80.

```
requestToken = client.GetRequestToken(
                    callback: "oob" //Optional
                    );
```

Figura 4-80 Código para la recuperación de *Tokens* temporales, en *Example_OAuth*

Como se avisa por consola, no tienen que proporcionarse los *RequestTokens* ya que se guardan en la librería al finalizar la operación anterior. Por lo que se prosigue con el último paso del proceso, con el código de la Figura 4-81.

```
        var verification = Console.ReadLine();
…
accessToken = client.GetAccessToken(
                    oauthVerifier:verification, //MANDATORY
                    requestToken:null, //Optional
                    requestSecret:null //Optional
                    );
```

Figura 4-81 Código para la recuperación de *Tokens* definitivos, en *Example_OAuth*

Además del código modificado previa ejecución de la prueba, durante esta se ha de introducir un número de verificación que ofrece el portal de *Connect* durante el proceso de autorización.

La captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-82 , Figura 4-83 y Figura 4-84.

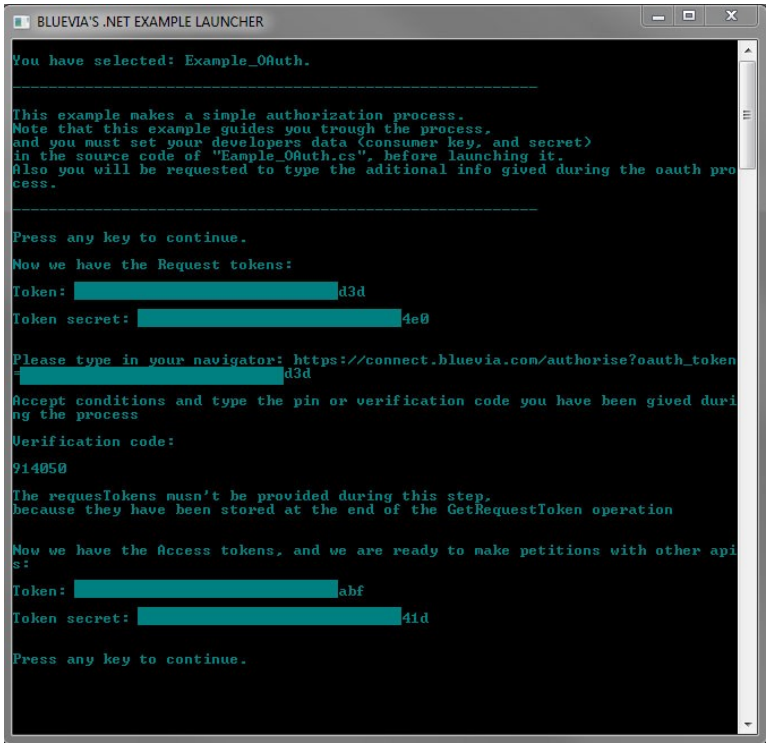


Figura 4-82 Captura de la consola al ejecutar ***Example_OAuth***

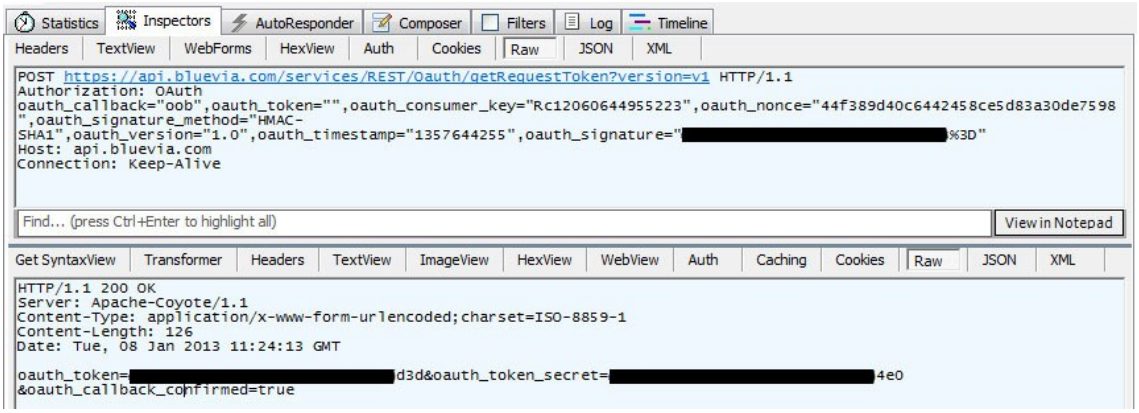


Figura 4-83 Captura de las trazas de red de la petición de *Tokens* temporales

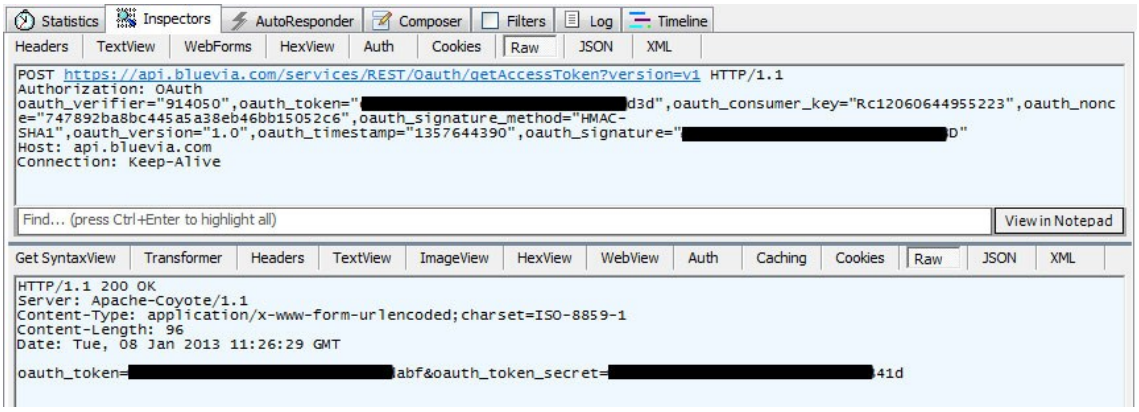


Figura 4-84 Captura de las trazas de red de la petición de *Tokens* definitivos

4.4.6 Validación de *Payment*

- **A través de *Example_Payment:***

Nota: En el momento de escribir esta parte de la memoria, ya se habían incorporado los servicios de la versión 2 a la plataforma de *Bluevia,* que no eran directamente compatibles con lo tratado en este *PFC*. Por lo que a pesar de que la versión 1.6 seguía activa, no se podían realizar nuevas autorizaciones para los modos *Sandbox* o *Test,* ni crear nuevas aplicaciones de dicha versión; haciendo imposible la demostración de la validación del API de *Payment.*
No obstante, se mostrarán las peticiones fallidas para demostrar la corrección del formato de las peticiones.

En el ejemplo se realiza un proceso de *OAuth* para recuperar unos *Tokens* que simbolicen el permiso de un usuario para realizar una transacción. El código necesita inserción de datos por parte del usuario que pretenda ejecutarlo; ya que este servicio necesita información delicada -y dichos datos de la aplicación de pruebas no pueden ser compartidos-. Este proceso de *OAuth* debe definir los datos de la transacción que va a ser realizada, como puede verse en el código de invocación de la Figura 4-85.

```
string yourConsumerKey = "Gy12012656******";
string yourConsumerSecret = "************";
uint yourAmount = 0;
String yourCurrency = "EUR";
String yourName = "bluevia";
String yourServiceID = "45b5f2517b8af797de******";

requestToken = client.GetPaymentRequestToken(
                    amount: yourAmount, //MANDATORY
                    currency: yourCurrency, //MANDATORY
                    name: yourName, //MANDATORY
                    serviceID: yourServiceID, //MANDATORY
                    callback: "oob" //Optional
                    );
…
var verification = Console.ReadLine();
…
accessToken = client.GetPaymentAccessToken(
                    oauthVerifier: verification, //MANDATORY
                    );
```

Figura 4-85 Código para autorizar la transacción de *Example_Payment*

Se realiza la obtención de *Tokens* temporales como se aprecia en el tráfico de la Figura 4-86.

Figura 4-86 Captura de las trazas de red de la petición de *Tokens* temporales para autorizar una transacción

Pero no puede obtenerse el código de verificación en *Connect* para seguir con la autorización, como se aprecia el la captura de la página de autorización en la Figura 4-87.

Figura 4-87 Captura del fallo en la autorización de la transacción en *Connect*

Se muestra el resto del código de la prueba, para completar la descripción, en las figuras: Figura 4-88, Figura 4-90 y Figura 4-91.

```
paymentResult = client.Payment(
                    amount: yourAmount, //MANDATORY
                    currency: yourCurrency, //MANDATORY
                    endpoint: null, //Optional
                    correlator: null //Optional
                    );
```

Figura 4-88 Código para ejecutar la transacción de *Example_Payment*

A pesar de no poder autorizar transacciones, se realiza una petición de pago para poder ilustrar el correcto formato de esta. En la Figura 4-89 puede observarse el cuerpo XML-RPC, y

el requisito de llevar en la cabecera de *OAuth* el mismo *TimeStamp* en segundos -contados desde el 1 de Enero de 1970-, que el pasado como parámetro RPC en el cuerpo del mensaje.

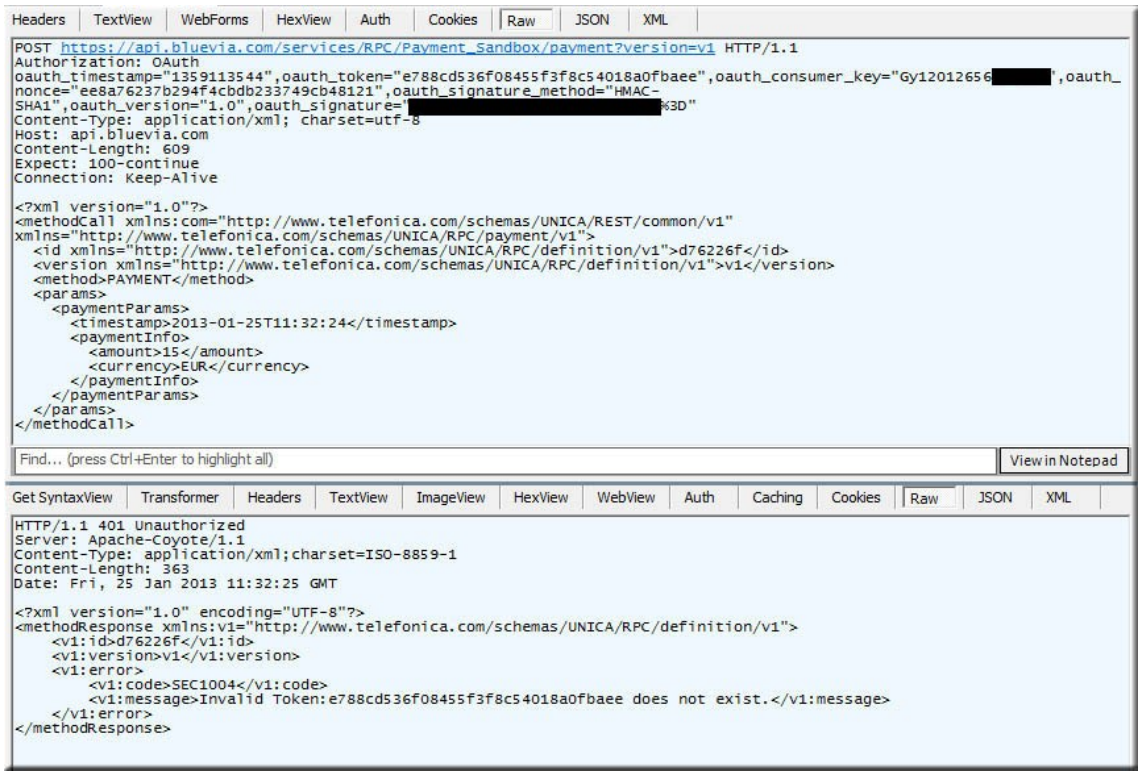


Figura 4-89 Captura del tráfico generado durante una petición de pago de transacción

```
paymentStatus = client.GetPaymentStatus(
                    transactionId: paymentResult.transactionId //MANDATORY
                    );
```

Figura 4-90 Código para recuperar el estado de la transacción de *Example_Payment*

```
var cancelResult = client.CancelAuthorization();
```

Figura 4-91 Código para cancelar la autorización de la transacción de *Example_Payment*

4.4.7 Validación de SMS

- **A través de *Example_SMS_MT:***

En este ejemplo de forma similar al caso de MMS, se puede observar el envío de un SMS y la recuperación de su estado. La validación se realiza entonces en dos partes:

La primera con el envío del mensaje, como se ilustra en la Figura 4-92.

```
string statusId = client.Send(
                    destination: "54666112233", //MANDATORY
                    text: "SANDBLUEDEMOS This is a Dummie SMS for SMS_MT", //MANDATORY
                    endpoint: null, //Optional
                    correlator: null //Optional
                     );
```

Figura 4-92 Código del envío de un SMS con un cliente SMS_MT, de *Example_SMS_MT*

Y la segunda con la recuperación del estado del envío con el código de la Figura 4-93.

```
deliveryInfos = client.GetDeliveryStatus(
                    messageId: statusId //MANDATORY
                    );
```

Figura 4-93 Código de recuperación del estado de envío del SMS, de *Example_SMS_MT*

La captura de pantalla de la aplicación y el tráfico de red aparecen en las figuras: Figura 4-94, Figura 4-95 y Figura 4-96.

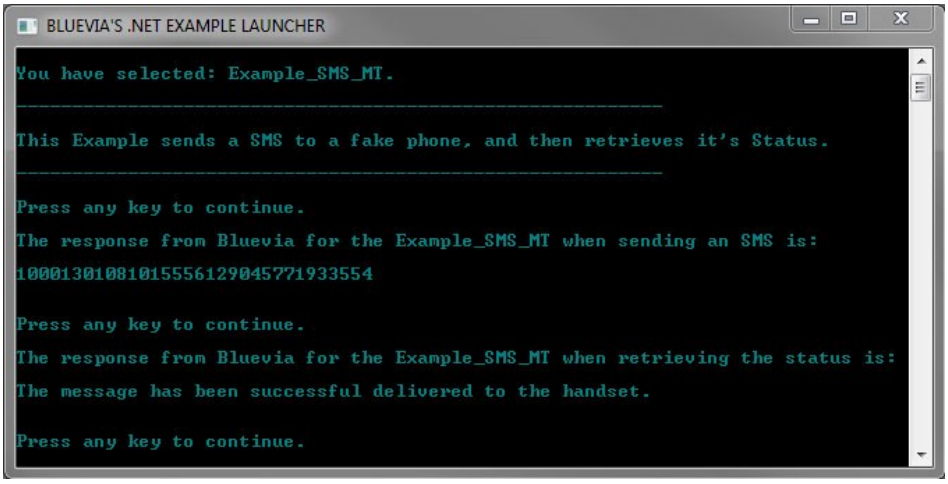


Figura 4-94 Captura de la consola al ejecutar *Example_SMS_MT*

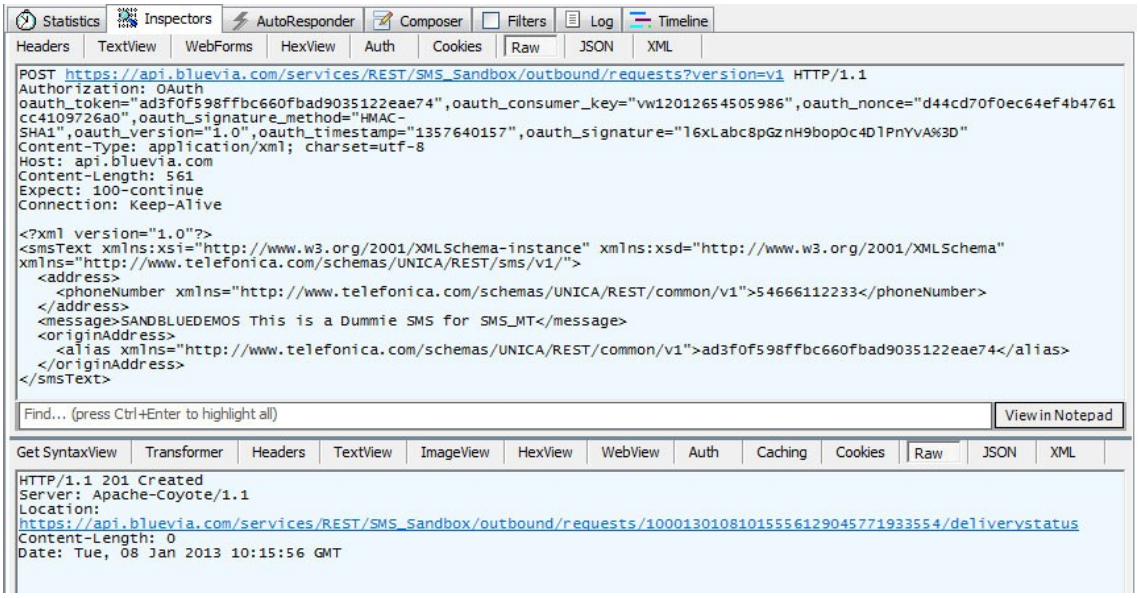


Figura 4-95 Captura de las trazas de red al enviar un SMS

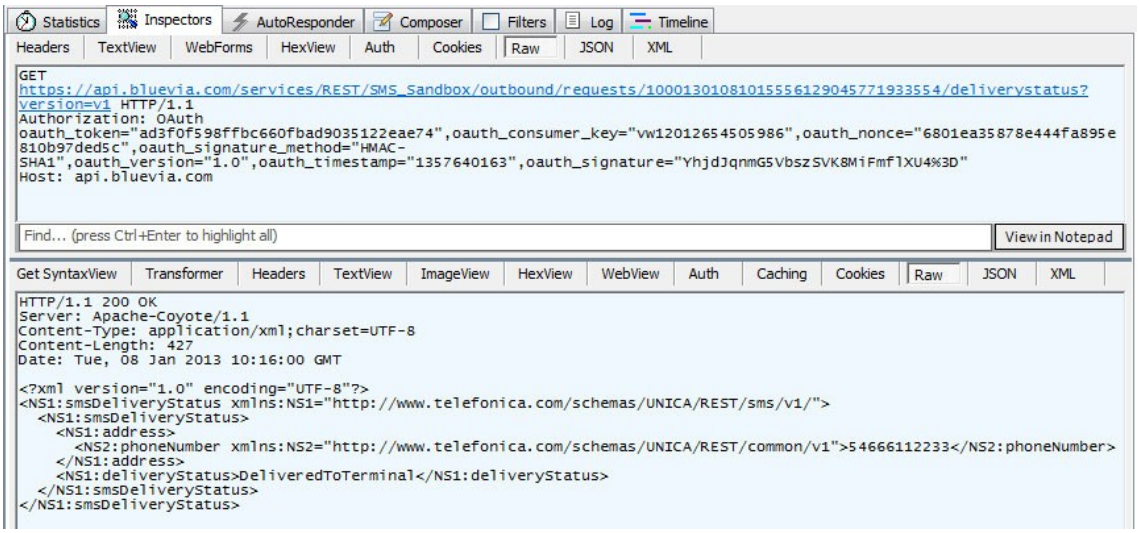


Figura 4-96 Captura de las trazas de red al recuperar el estado del envío de un SMS

En este caso, además de comprobar la invocación de los servicios, se ha considerado la codificación correcta de los caracteres en el mensaje. Para ello se realiza la prueba en modo *LIVE* para enviar el mensaje a varios dispositivos móviles de diferentes países donde existan caracteres distintos a los del alfabeto anglosajón, resultando exitosa al visualizarse correctamente los caracteres en los dispositivos.

- **A través de *Example_SMS_MO:***

Esta prueba envía un SMS al número corto de un buzón y luego descarga la lista de mensajes de dicho buzón.

El primero de los pasos, salvo en el valor del parámetro *destination* -que en este caso hace referencia al número corto de un buzón de pruebas-, es idéntico al envío de SMS mostrado en **SMS MT**, por lo que se expondrá el código de invocación, pero no la captura del mensaje de red. El paso de recuperación de la lista de mensajes, si se expone en su totalidad.

Puede observarse que el código de envío del mensaje solo varía en el número de *destination*, como se aprecia en la Figura 4-97. Y se recuperan los mensajes del buzón con el código de la Figura 4-98.

```
clientMT.Send(
                    destination: "546780", //MANDATORY
                    text: "SANDBLUEDEMOS This is a Dummie SMS for SMS_MO", //MANDATORY
                    endpoint: null, //Optional
                    correlator: null //Optional
                     );
```

Figura 4-97 Código del envío de un SMS con un cliente SMS_MT, de *Example_SMS_MO*

```
inboxMessages = clientMO.GetAllMessages(
                    registrationId: "546780" //MANDATORY
                    );
```

Figura 4-98 Código dela recuperación de todos los SMS de un buzón, de *Example_SMS_MO*

La captura de pantalla de la aplicación aparece en la Figura 4-99.

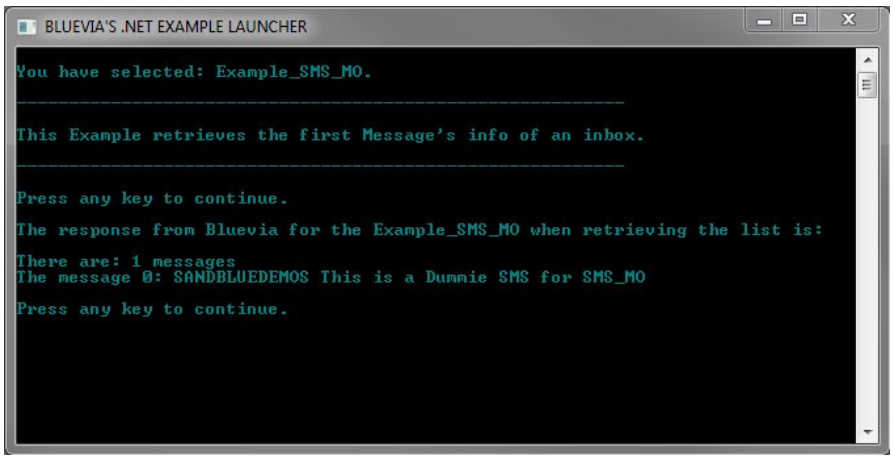


Figura 4-99 Captura de la consola al ejecutar *Example_SMS_MO*

En la figura Figura 4-100 se presenta la lista de SMS -con el único mensaje enviado previamente-.

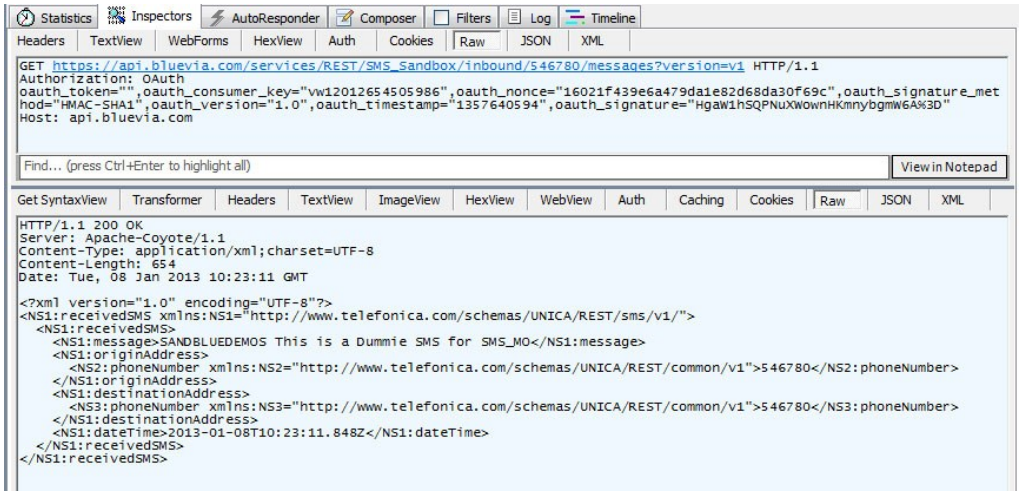


Figura 4-100 Captura de las trazas de red al recuperar los mensajes del buzón

En **SMS MO**, además de las pruebas para los diferentes modos, y con diferentes parámetros, también se ha considerado la recuperación correcta de los caracteres de los mensajes. Este requisito se ha validado descargando y visualizando mensajes desde buzones de diferentes países, enviados tanto desde la librería, como desde dispositivos móviles.

- **A través de *Example_SMS_Notifications:***

Salvo la URL que invoca la librería, todo el proceso y parámetros de la prueba son idénticos a los de ***Example_MMS_Notifications*** por lo que no se repetirán aquí.

4.4.8 Validación de control y captura de Errores

Además de las pruebas para comprobar la invocación y correcto funcionamiento de los servicios, se han realizado pruebas para comprobar que saltasen las excepciones correspondientes atendiendo a las especificaciones de cada API.

Por ejemplo, en el caso del número de teléfono de destino en el servicio de notificaciones, que está obligado a ser un *String* numérico.

Si es nulo, vacío o rellenado con caracteres no numéricos, como los aparecidos en el código de la Figura 4-101, la librería devuelve la excepción que se observa en la pantalla de la Figura 4-102.

```
string youPhoneNumber = "NoNumericString";
string yourCorrelator = "A Correlator";
string yourEndpoint = "http://anyEndpoint";
string yourCriteria = "A Criteria";
```

Figura 4-101 Código de valores erróneos

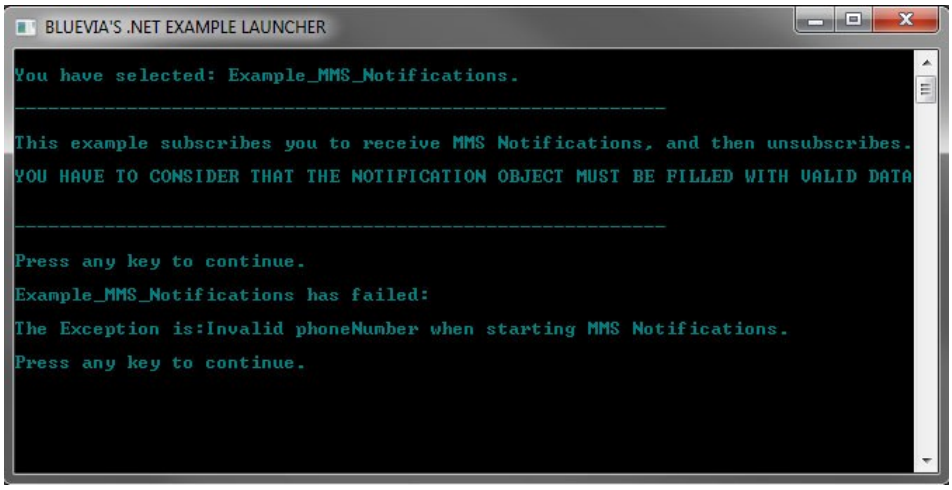


Figura 4-102 Captura de pantalla de un error controlado por la librería, durante la ejecución de la aplicación de demos.

4.5 GENERACIÓN DE LA DOCUMENTACIÓN

Como se describió en el apartado de diseño, uno de los requisitos para el nuevo SDK, heredado de las versiones anteriores, era el de ofrecer una documentación adecuada del paquete:

- Las clases públicas del código deben de llevar la documentación necesaria para poder ser abordadas rápidamente.
- Además, debe de crearse una guía o manual explicando de forma general el funcionamiento de *Bluevia*, y el uso de los diferentes servicios ofrecidos por el SDK.

La resolución de estos requisitos se cristaliza en un manual de uso del SDK, generado con ***Doxygen***, que engloba toda la documentación relativa al uso de la librería: descripción de clases, guías de uso de las API y guías de uso del SDK.

Nota: **A pesar de que inicialmente el mercado potencial del SDK se reparte en varios idiomas; para procurar un mayor alcance en este aspecto, y dado que los recursos para este punto eran limitados, el proyecto se realizará al completo en inglés.**

4.5.1 Documentación del código y referencias de clase

Desde la jefatura del grupo de SDKs, se demanda que todas las clases que hagan de interfaz de cara al desarrollador, estén adecuadamente comentadas; de forma que puedan generarse referencias de clase a partir de ellas.

Para ello se siguen en parte los consejos de ***Microsoft*** sobre el estilo para el desarrollo en ***C#***; buscando un compromiso con los requisitos de ***Doxygen*** para poder generar correctamente las referencias.

De modo que salvo las clases contenedoras de información: ***Schemas***, autogeneradas a partir de los documentos XSD proporcionados por los diseñadores de *Bluevia*; todas las clases de la librería contienen:

- Una cabecera informativa de *Bluevia*, de acuerdo al requerimiento de la dirección del proyecto, como se muestra en la Figura 4-103.

```
// ------------------------------------------------------------------------ //
// BlueVia is a global iniciative of Telefonica delivered by Movistar and O2. //
// Please, check out www.bluevia.com and if you need more information contact //
// us at support@bluevia.com"                                               //
// ------------------------------------------------------------------------ //
```

Figura 4-103 Cabecera informativa de *Bluevia* insertada como comentario en todas las clases no autogeneradas de la librería

- Una introducción a la clase -recomendada por ***Microsoft***- con dos etiquetas:
  - *<copyright file="clase.cs" company ="Telefonica R&D">,* relativa a la propiedad del documento.

- *<summary>*, con una pequeña descripción.

- Una etiqueta *<value>* -recomendada por ***Microsoft***- para cada atributo de la clase, describiendo su utilidad.

- Un bloque para cada método con las etiquetas -recomendadas por ***Microsoft***-:

  - *<summary>*, que contenga una descripción de la funcionalidad del método.

  - *<param name = "nombre">*, para cada parámetro, con la descripción de este.

  - *<returns>*, con el valor devuelto por el método si procede.

- Para facilitar futuras modificaciones, en algunas clases se crean comentarios adicionales para explicar, justificar o resaltar el funcionamiento de ciertas partes del código, como se muestra en la Figura 4-104.

```
//The content-type header must be set with the following instruction ...
request.ContentType = contentType;

//... so deleting from the headerCollection, to avoid an exception ...
headerCollection.Remove(HttpTools.ContetTypeKey);
//... when the whole collection of headers is set.
request.Headers.Add(headerCollection);
```

Figura 4-104 Ejemplo de comentarios justificando sentencias de código

O enlaces a recursos online, , como se muestra en la Figura 4-105.

```
Must follow <a href="http://www.iso.org/iso/country_codes.htm)">ISO-3166</a>
```

Figura 4-105 Ejemplo de comentario enlazando a un recurso online

Con estas cabeceras y etiquetas se generan de forma automática referencias HTML con la descripción de cada clase, que formarán parte del manual de uso del SDK.

***Nota*: Se presenta una muestra de las páginas generadas en el anexo: 7.9.**

4.5.2 Guías de uso de las API

Para el adecuado uso del SDK, se crean guías con la descripción del funcionamiento de cada uno de los servicios: *OAuth*, *Advertising*, *Directory*, *Location*, MMS, *Payment* y SMS.

Estas guías contienen una introducción al servicio ofrecido por *Bluevia,* y una descripción pormenorizada de los elementos del SDK disponibles para acceder a este.

Las guías están compuestas por:

- **Nombre del API y descripción del servicio**. Se presenta una breve introducción al servicio, con enlaces a la descripción general ofrecida en *Portal*; y si procede enlaces a otros recursos online, o partes del manual del SDK.

- **Índice de la Guía**. Con enlaces a cada una de las partes para una navegación más sencilla.

- **Descripción de los clientes del API**. Se enumeran los clientes disponibles del API, con los requisitos especiales que puedan necesitar.

  Se presentan enlaces a la descripción generada a partir del código fuente de estas clases. Y también los fragmentos de código necesarios para instanciar estos clientes.

- **Descripción de los Métodos del API**. Se enumeran los métodos disponibles en el API. Cada uno de ellos con una descripción de su funcionamiento, requisitos, parámetros y el fragmento de código necesario para invocarlos.

  También se muestra el enlace a la descripción generada por el código fuente de estos.

***Nota*: Se presenta una muestra de las guías creadas en el anexo: 7.9.1.2 Guías de uso de los servicios**

4.5.3 Guías de uso del SDK

Por último, para completar el manual de uso del SDK, se demandan una serie de guías que ayuden al desarrollador a utilizar y comprender el SDK en aspectos más generales que los cubiertos por los puntos anteriores.

Los requisitos que deben cubrirse son:

- Describir el contenido del SDK y sus requisitos para hacerlo funcionar.

- Describir de forma abreviada la plataforma de *Bluevia* y los requisitos para acceder a ella.

Para ello se crean cuatro guías más

- **Introducción:** Esta guía ofrece una breve presentación de *Bluevia* y los servicios que esta ofrece, mostrando enlaces a los recursos online que describen las funcionalidades de esta plataforma en *Portal*.

  También lista el contenido del paquete del SDK, describe la estructura de la librería, y la funcionalidad general que esta ofrece; además de enlazar el contenido del resto del manual.

- **Guía de uso de *Bluevia*:** Esta guía describe brevemente el funcionamiento general de la plataforma, aportando enlaces a los recursos online que amplían la información en *Portal.*

  Resume los modos de acceso, los tipos de identificadores necesarios, y el proceso de autorización *OAuth*.

  Tras ello, describe ayudado de ejemplos de código, el proceso de invocación de un servicio usando los clientes del SDK, y la forma de capturar y analizar las posibles excepciones que surjan durante dicha invocación.

- **Guía de inicio del SDK:** En esta guía se describe el entorno necesario para poder utilizar la librería.

  Tras ello se ofrece una guía paso a paso, ilustrada con capturas de pantalla, y enlaces a recursos *online*; para:

  - Instalar la librería y usarla en un proyecto propio.
  - Comenzar un proyecto sobre el código fuente del SDK.
  - Ejecutar y depurar el proyecto.
  - Instalar y configurar ***Fiddler2*** para inspeccionar las llamadas.

- **Guía de las demos:** Esta última guía describe como cargar y lanzar los ejemplos contenidos en el paquete, e ilustra con imágenes como capturar el tráfico generado con ***Fiddler2***.

***Nota*: Se presenta una muestra de las guías creadas en el anexo: 7.9.1.3Manuales de uso del SDK**

4.5.4 Documentación del paquete de acceso privado

Al igual que para el caso público se ha generado documentación para el paquete *Trusted* de acceso privado, siguiendo el mismo procedimiento para su elaboración:

- Comentando el código.
- Generando un manual de uso del SDK, compuesto de referencias de clases, guías de uso de las API y Guías de uso del SDK.

Pero para este caso, además de variar con respecto al paquete de acceso libre en la diferencia de las referencias a los clientes finales de la librería -que como ha podido verse en los puntos de diseño e implementación, son las clases que difieren en ambas versiones-. La documentación cambia en el aspecto de la autorización y autenticación.

En la documentación del paquete de acceso privado:

- No existe guía de uso del API de *OAuth.*
- No se hace referencia al proceso de *OAuth*.
- En las guías de uso del SDK, se hace referencia a la configuración y uso de los certificados para conexiones SSL de dos vías; mostrando como configurar el entorno, los clientes, e incluso ***Fiddler2*** para monitorizar el tráfico a través de las conexiones generadas.

5 HISTORIA DEL PROYECTO

El grupo de trabajo de los SDKs de *Bluevia*, está compuesto por 5 desarrolladores -uno por cada lenguaje de SDK- y un jefe o coordinador, en Madrid. Su cometido es el de mantener y dar soporte a estos SDKs, además de realizar proyectos de desarrollo para el ámbito de *Bluevia*.

Los SDKs de *Bluevia* llegan a ***Telefónica I+D*** Madrid, en su versión 1.0, en Enero de 2011; y después de varios meses trabajando en su mantenimiento, comienza a plantearse por parte del grupo de desarrollo un deseo de "refactorizarlas" en profundidad. Esta idea se transmite a la dirección de *Bluevia* en la primavera del mismo año, y se recibe el permiso para acometerla en la segunda quincena de Enero del 2012, aprovechando el tiempo de desarrollo para la versión 1.6.

Durante las dos primeras semanas se leen las especificaciones requeridas y comienza a diseñarse la estructura principal, que será similar a la de las versiones anteriores del SDK de ***Android***.

Una vez especificado el nivel de acceso a la red, se comienza su implementación, y la especificación de los siguientes niveles. Este proceso lleva alrededor de mes y medio de trabajo. Durante este tiempo, y a pesar de llevar trabajando con ***C#*** varios meses, me documento sobre las opciones que ofrece el lenguaje para procurar realizar un código eficiente y sencillo.

La implementación comienza creando un proyecto enfocado a la conectividad en la "solución" de la versión 1.5 del SDK, para apoyarme en los recursos ya existentes. Las pruebas del desarrollo del primer nivel se realizan con peticiones de datos pregenerados de todos los formatos de mensaje.

La especificación del segundo nivel se completa en un par de días, mientras que la del nivel de inteligencia de API se esboza inicialmente, pero no se completa hasta los últimos días del desarrollo del código, ya que abarcaba la decisión en común de los múltiples parámetros de las interfaces de cara al usuario.

Dado que el diseño lo permitía, la implementación se realiza en bloques de funcionalidad enfocados a cubrir APIs concretos. El primero dada su relativa sencillez, y la cantidad de pruebas que proporcionaba, es el de *Advertising*, que una vez completo, verifica prácticamente la estructura general implementada.

El desarrollo a partir de ahí se realiza rápidamente debido a la generación automática de objetos mediante los XSDs proporcionados por la dirección de la plataforma, y a la homogeneidad en el funcionamiento general que permitía el diseño. Los ejemplos de la aplicación de demos se creaban a terminar los respectivos APIs.

A principios de Abril se reciben los nuevos requisitos para el acceso a los servicios privados, y mediados de ese mes, las librerías están completadas aunque sin probar completamente en los nuevos entornos, por lo que se comienzan las labores de finalizar la documentación del código y crear el manual.

La primera semana de Mayo, pueden empezar a probarse los servicios reales en los nuevos entornos, y surge el problema de la codificación de caracteres en la mensajería; pero se soluciona en una semana, y el paquete está listo para auditar alrededor del 10.

En Junio aproximadamente, ya fuera de ***Telefónica I+D*** comienza el proceso de elaborar esta memoria.

Durante un mes me documento con los recursos de la bibliografía, y especifico los puntos que deben tratarse, mientras que los meses posteriores voy desarrollando dichos puntos hasta su conclusión en Febrero de 2013.

En el periodo del desarrollo de la memoria, se han dado varios cambios en el funcionamiento de la plataforma *Bluevia.*

- Se añades varios SDKs no oficiales a la lista de los disponibles en ***GitHub.***
- Se incorpora soporte y productos ***Arduino*** en el *Portal.*
- Cambian las respuestas del API de *Directory,* desaparece la *AppStore* y la certificación de aplicaciones.
- Desde Noviembre la operadora de telefonía ***Telenor*** se suma a la plataforma.
- En Enero de 2013, se incorpora la versión 2 de la plataforma, haciendo imposible crear nuevas aplicaciones para la versión 1.6, aunque manteniendo el soporte para las ya creadas.

6 CONCLUSIONES

6.1 Objetivos logrados

Puede concluirse que se ha realizado el objetivo principal de este proyecto, al haber implementado unas librerías en ***C#*** de acceso a los servicios de *Bluevia 1.6,* que simplifican bastante esta tarea a los desarrolladores.

Estas librerías podrán descargarse de la web[6], aún durante los primeros meses de 2013.

Se ha cumplido también con los requisitos de ofrecer un código adecuadamente documentado, junto con guías completas de uso, y una aplicación de demostración para todos los APIS.

Además, se aúna la estructura y uso con la de los SDKs del resto de lenguajes -con un diseño que permite una fácil escalabilidad-, al estar compuestas las librerías por los mismos elementos principales, y ofrecer la misma interfaz.

Todo esto se ha realizado dentro de los plazos establecidos -ampliados en un par de ocasiones-, estando el paquete de entrega, ya auditado semanas antes del lanzamiento a producción de la versión 1.6 de la plataforma.

Por último, se han podido satisfacer los deseos informales, basados en la experiencia previa en el desarrollo de las versiones anteriores del SDK, de ofrecer un flujo de funcionamiento más lineal y depurable en vistas al mantenimiento futuro del SDK; eliminando la dependencia en las librerías externas, y aligerando la carga del *framework*, haciendo necesario tan solo la versión ***Client profile***; factores que hacen que se ejecuten ligeramente más rápido los servicios que en versiones anteriores de las librerías.

6.2 Líneas futuras de trabajo

A pesar de que el SDK está en producción, se podría mejorar de las siguientes formas:

- **Demos en ASP *.NET*:** Con la atención que empiezan a acaparar los nuevos estándares de la web -la combinación HTML5, CSS3 y JavaScript- debido al potencial que ofrecen, y usando las librerías ya creadas; podría ofrecerse un nuevo paquete extra de aplicaciones de demostración, que corriesen esta vez sobre un servidor IIS. De modo que los desarrolladores pudiesen acceder rápidamente a ejemplos de integración de las librerías con páginas web.

- **Funcionalidad para formato de mensajes en JSON:** También sobre la estructura existente, podría dotarse al SDK de la posibilidad de elegir el formato de los cuerpos de los mensajes en el caso de las comunicaciones SOAP o RPC, y permitir así el envío más ligero de JSON en vez de XML.

Para ello habría que crear un "serializador" y un "parseador" JSON. La librería *System.Runtime.Serialization.Json.DataContractJsonSerializer* funciona prácticamente de la misma forma que *System.Xml.Serialization.XmlSerializer*, por lo que se podrían realizar estas clases reutilizando la estructura de las de XML.

Por otro lado, habría que añadir un parámetro de elección del formato del formato deseado en los métodos de invocación de los servicios que usasen este tipo de comunicación.

- **APIs V2:** Desde Enero de 2013 *Bluevia* ha puesto a disposición del público sus APIs V2, poniendo fecha de caducidad a las aún existentes APIs 1.6.

  Con esta nueva versión de la plataforma, desaparecen varios de los servicios, y los que permanecen, difieren ligeramente de los anteriores.

  - Cambian los *endpoints*, lo que supondría modificar los valores para la creación de estos en los archivos **Constants.cs**.

  - Y cambia el sistema de autorización a *OAuth*2.0, por lo que habría que cambiar el método *Authenticate* del ***BV_Connector***, y la clase ***OAuthManager***; adecuando además el resto de la estructura de clientes a los nuevos parámetros exigidos.

- **Portar las librerías a Windows Phone7 y Mono:** En un momento en que los *smartphones* se han convertido en una parte importante del mercado de las telecomunicaciones, los productos para ellos son una pieza clave del negocio; por lo que *Bluevia* ha apostado directamente por SDKs como el de ***Android***, y ha recibido bien las colaboraciones de otros como el de ***Objetive-C*** para sistemas ***IPhone***[47].

  Sin embargo el sistema operativo para dispositivos móviles de ***Microsoft***, queda huérfano de librerías de *Bluevia*, ya que a pesar de lo que podría suponerse en primera instancia, las librerías creadas en este proyecto no pueden emplearse directamente en ***Windows Phone7*** debido a que este no soporta toda la funcionalidad del ***.NET*** *Framework* 4.

  Empezando simplemente por las funciones de codificación de caracteres, habría que modificar muchas de las herramientas usadas en el SDK; aunque podría mantenerse la estructura creada, y gran parte del código para portar las librerías ***.NET*** de *Bluevia* al sistema operativo móvil.

  De la misma forma, probablemente fuese necesario cambiar el uso de algunas funciones para poder correr las librerías en entornos del proyecto ***Mono***, lo que proporcionaría la posibilidad de usarlas en más sistemas operativos y dispositivos.

7 ANEXOS

En este bloque se desarrollan puntos que complementan la información de la memoria.

- En el punto 7.1 se presenta un presupuesto estimado del proyecto.
- En el punto 7.2 se enumeran las especificaciones particulares para el acceso a los servicios de *Bluevia.*
- En el punto 7.3 se realiza una descripción más detallada que la cubierta en el estado del arte, de los servicios ofrecidos por los APIs de *Bluevia.*
- En el punto 7.4 se muestra una pequeña guía de instalación del entorno necesario para ejecutar las librerías del SDK.
- En el punto 7.5 se realiza una breve descripción de los proxies de monitorización de red utilizados durante el desarrollo, y una pequeña guía de uso de ***Fiddler2.***
- En el punto 7.6 se presenta la estructura de archivos en la que se estructura el repositorio del proyecto de desarrollo del SDK de ***C#.***
- En el punto 7.7 se presenta una guía para la creación y uso de una cuenta de desarrollador de *Bluevia.*
- En el punto 7.8 se presenta una guía para la creación y uso de una cuenta de usuario final de *Bluevia.*
- En el punto 7.9 se presenta una breve descripción de la aplicación de documentación ***Doxygen***, y un ejemplo simbólico de la documentación generada en el proyecto.
- El punto 7.10 es un glosario de parte de la terminología usada en la memoria.
- El punto 7.11 es un listado con los recursos referenciados en esta memoria.
- El punto 7.12 es una bibliografía de los recursos consultados durante el desarrollo del proyecto, y la memoria.

7.1 PRESUPUESTO

En este apéndice se presentan justificados los costes globales de la realización de este proyecto. Tales costes, imputables a gastos de personal, y material se pueden deducir de la siguiente tabla.
Las cifras están basadas en el presupuesto de otro PFC[53] también realizado en ***Telefónica I+D.***

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1. **Autor**: Isidro Gómez Crisóstomo.
2. **Departamento**: Telemática.
3. **Descripción del Proyecto**:
   - -Título: Exposición de APIs de red de operador, a desarrolladores, basada en orientación a objetos
   - -Duración en meses (m) 4(desarrollo), 3(memoria)
   - -Tasa de costes indirectos: 20%
4. **Presupuesto total del Proyecto** (en Euros):
5. **Desglose presupuestario** (costes directos):

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Meses	Coste mes	Coste (euros)
Gómez Crisóstomo, Isidro	XXXX	Ingeniero Junior	7 m	3.000,00	21000,00
Vélez Tarilonte, Enrique	XXXX	Jefe de Proyecto	0.8 (4m/5proy)	6.000,00	4.800.00
Basanta Val, Pablo	XXXX	Ingeniero Senior	1m	4.500,00	4.500,00
4x integrantes del grupo de SDKs	XXXX	Ingeniero junior	0.5m (diseño común)	3.000,00	6000.00
				Total	***36.300.00***

EQUIPOS

Descripción	Coste (euros)	%Uso dedicado	Dedicación	Periodo de depreciación	Coste Imputable
Pc portátil	900	100	3	60	45,00
Pc sobremesa	600	100	4	60	40,00
2x Monitores	220	100	4	60	14,67
Ratón	10	100	7	60	1,17
Teclado	10	100	7	60	1,17
Cascos	10	100	7	60	1,17
				Total	***104,77***

OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Empresa	Costes Imputables
Licencias Software	Telefónica I+D	150.00
	Total	***150.00***

6. **Resumen de costes**

	Presupuesto Costes Totales
Personal	36.300
Amortización	105
Costes de funcionamiento	150
Costes indirectos	7.311
Total	***43.866***

Tabla 1 Presupuesto

7.2 RESUMEN DE LAS ESPECIFICACIONES PARA EL ACCESO A LOS SERVICIOS DE BLUEVIA

A continuación se ofrecen varias tablas-resumen -Figura 7-1, Figura 7-2, Figura 7-3, Figura 7-4, Figura 7-5, Figura 7-6, Figura 7-7- con los aspectos más relevantes de las especificaciones de cada servicio de acceso público de las API ofrecidas por la plataforma. En ellas se considera:

- Que la conexión y comunicación para todos ellos, se realiza mediante HTTP sobre SSL a un *endpoint* principal.
- Que los datos identificadores de la aplicación son obligatorios en todos los casos *2-3legged*, y que los datos identificadores de usuario son obligatorios para los casos *3legged*.
- Que los datos referentes a los servicios -tanto obligatorios como opcionales-, son objetos y campos especificados públicamente, tanto en la definición de los servicios ([1]), como en los archivos XSD que acompañan al código fuente del proyecto.

Recordemos que el caso de OAuth *2legged* en *Bluevia* -como se pudo ver en el punto 2.1.4- hace referencia a todas aquellas aplicaciones que no necesitan de una autorización del usuario para poder acceder a la plataforma, mientras que hace referencia a aquellas aplicaciones que si necesitan dicha autorización.

Nota: Los nombres de los servicios aquí listados son directamente lo de los endpoints respectivos.

Nombre del servicio ofrecido por el API de *OAuth*	**Modos operativos soportados**	**Datos obligatorios de la petición**	**Datos opcionales de la petición**	**Formato de la Petición**	**Formato de la Respuesta**
`getAccessToken`	*-2legged* -LIVE, TEST (no válido para autorizar Payment), SANDBOX.	-Cabecera extra de OAuth. -Cabeceras extra de OAuth, referentes al servicio y modo y un Objeto de la transacción (cantidad, divisa, nombre de producto, identificador del servicio), si la operación se	-Dirección de callback. -MSISDN si se pretende autorizar la aplicación vía teléfono móvil (sólo válido para modo LIVE, y no válido para autorizar Payment).	-Operación POST HTTP. -Cabeceras extra de OAuth, en la cabecera de Autorización de HTTP. -Objeto de la transacción empaquetado en formato FormUrlEncoded en el cuerpo.	-Objeto de la respuesta empaquetado en FormUrlEncoded.

		usa para autenticar una operación de Payment.			
getRequestToken	*-3legged* -LIVE, TEST (no válido para autorizar Payment), SANDBOX.	-Cabeceras extra de OAuth. -Código de verificación		-Operación POST HTTP. -Cabeceras extra de OAuth, en la cabecera de Autorización de HTTP. -Código de verificación, en la cabecera de Autorización de HTTP.	-Objeto de la respuesta empaquetado en FormUrlEncoded.

Figura 7-1 Tabla de requisitos exigidos por el servicio de OAuth.

Nombre del servicio ofrecido por el API de *Advertising*	**Modos operativos soportados**	**Datos obligatorios de la petición**	**Datos opcionales de la petición**	**Formato de la Petición**	**Formato de la Respuesta**
simple/requests	*2-3legged* LIVE, TEST, SANDBOX	-Código de espacio de anuncios. -Código de país, si la petición es *2legged*.	-Varios datos referentes a la selección de anuncios: política de madurez, palabras clave…	-Operación POST HTTP. -Objeto de la selección del anuncio empaquetado en formato FormUrlEncoded en el cuerpo.	-Objeto de la respuesta empaquetado en XML o JSON.

Figura 7-2 Tabla de requisitos exigidos por el servicio de *Advertising*.

Nombre del servicio ofrecido por el API de *Directory*	**Modos operativos soportados**	**Datos obligatorios de la petición**	**Datos opcionales de la petición**	**Formato de la Petición**	**Formato de la Respuesta**
UserInfo	*3legged* LIVE, TEST, SANDBOX	-Identificación de usuario objetivo.	-Al menos dos de los sets de datos disponibles (*Access, Personal, Profile, Terminal)*.	-Operación GET HTTP. -La URL se monta con el identificador del usuario objetivo. -Los datos opcionales viajan como	-Objeto de la respuesta empaquetado en XML O JSON.
UserAccessInfo			-Al menos		
UserPersonalInfo					

UserProfile UserTerminalInfo			uno de los campos de información disponibles del set en cuestión.	parámetros en la propia URL de la petición.	

Figura 7-3 Tabla de requisitos exigidos por el servicio de *Directory*.

Nombre del servicio ofrecido por el API de *Location*	Modos operativos soportados	Datos obligatorios de la petición	Datos opcionales de la petición	Formato de la Petición	Formato de la Respuesta
TerminalLocation	*3legged* LIVE, TEST, SANDBOX	-Identificación de usuario objetivo.	-Precisión útil requerida	-Operación GET HTTP. -Los datos viajan como parámetros en la propia url de la petición.	-Objeto de la respuesta empaquetado en XML O JSON.

Figura 7-4 Tabla de requisitos exigidos por el servicio de *Location*.

Nombre del servicio ofrecido por el API de SMS	Modos operativos soportados	Datos obligatorios de la petición	Datos opcionales de la petición	Formato de la Petición	Formato de la Respuesta
requests	*3legged* LIVE, TEST, SANDBOX	-Lista de destinos. -Mensaje.	-Dirección del servidor propio de notificaciones, más un identificador.	-Operación POST HTTP. -Objeto de la información del SMS empaquetado en formato XML o JSON en el cuerpo.	-Información de la respuesta, en la cabecera *Location* de HTTP.
deliverystatus		-identificador del mensaje enviado.		-Operación GET HTTP. -La URL se monta con el identificador del SMS enviado.	-Objeto de la respuesta empaquetado en XML o JSON.
messages	*2legged* LIVE, TEST, SANDBOX	-Código corto del servicio de buzones del país.		-Operación GET HTTP. -La URL se monta con el identificador del servicio de buzones del país.	-Objeto de la respuesta empaquetado en XML o JSON.
subscriptions		*Para subscribirse: -Código corto del servicio de		*Para subscribirse: -Operación POST HTTP.	

		buzones del país. -Dirección del servidor propio de notificaciones, más un identificador. -Palabra clave del servicio. *Para dar de baja: -El identificador de la subscripción.		-Objeto de la subscripción empaquetado en formato XML o JSON en el cuerpo. *Para dar de baja: -Operación DELETE HTTP. -La URL se monta con el identificador de las subscripción.	

Figura 7-5 Tabla de requisitos exigidos por el servicio de SMS.

Nombre del servicio ofrecido por el API de MMS	Modos operativos soportados	Datos obligatorios de la petición	Datos opcionales de la petición	Formato de la Petición	Formato de la Respuesta
`requests`	*3legged* LIVE, TEST, SANDBOX	-Lista de destinos. -Motivo del mensaje.	-Lista de archivos binarios que se desean adjuntar. -Dirección del servidor propio de notificaciones, más un identificador.	-Operación POST HTTP. -Objeto de la información del MMS empaquetado en formato XML o JSON en el cuerpo.	-Información de la respuesta, en la cabecera *Location* de HTTP.
`deliverystatus`		-identificador del mensaje enviado.		-Operación GET HTTP. -La URL se monta con el identificador del MMS enviado.	-Objeto de la respuesta empaquetado en XML o JSON.
`messages`	*2legged* LIVE, TEST, SANDBOX	-Código corto del servicio de buzones del país.	-Si a posteriori se desea descargar un archivo adjunto en particular con la operación `attachments`, deberá especificarse en esta operación con un parámetro extra.	-Operación GET HTTP. -La URL se monta con el identificador del servicio de buzones del país. -El parámetro extra, viajará en la propia URL de la petición.	-Objeto de la respuesta empaquetado en XML o JSON.
`messages/Id`		-Código corto del		-Operación GET	-Objeto de la

		servicio de buzones del país. -Identificador del mensaje deseado.		HTTP. -La URL se monta con el identificador del servicio de buzones del país y el identificador del mensaje.	respuesta empaquetado en XML o JSON.
attachments		-Código corto del servicio de buzones del país. -Identificador del mensaje deseado. -Identificador del archivo adjunto.		-Operación GET HTTP. -La URL se monta con el identificador del servicio de buzones del país, el identificador del mensaje y el identificador del archivo adjunto.	-Objeto de la respuesta empaquetado en XML o JSON.
subscriptions		*Para subscribirse: -Código corto del servicio de buzones del país. -Dirección del servidor propio de notificaciones, más un identificador. -Palabra clave del servicio. *Para dar de baja: -El identificador de la subscripción.		*Para subscribirse: -Operación POST HTTP. -Objeto de la subscripción empaquetado en formato XML o JSON en el cuerpo. *Para dar de baja: -Operación DELETE HTTP. -La URL se monta con el identificador de las subscripción.	

Figura 7-6 Tabla de requisitos exigidos por el servicio de MMS.

Nombre del servicio ofrecido por el API de *Payment*	**Modos operativos soportados**	**Datos obligatorios de la petición**	**Datos opcionales de la petición**	**Formato de la Petición**	**Formato de la Respuesta**
payment	*3legged* LIVE, SANDBOX	-Un Objeto de la transacción (cantidad, divisa, *timestamp*) encapsulado en un objeto RPC que define la operación.	-Dirección del servidor propio de notificaciones, más un identificador.	-Operación RPC sobre un POST de HTTP, con el mismo *timestamp* que el contenido en el objeto	-Objeto de la respuesta RPC empaquetado en XML o JSON.

				interno. -Objeto de la operación encapsulado en XML o JSON.	
getPaymentStatus		-Identificador de la transacción encapsulado en objeto RPC que define la operación.		-Operación RPC sobre un POST de HTTP. -Objeto de la operación encapsulado en XML o JSON.	
cancelAuthorization		-Objeto RPC que define la operación.			

Figura 7-7 Tabla de requisitos exigidos por el servicio de *Payment*.

7.3 DESCRIPCIÓN DE LOS SERVICIOS OFRECIDOS POR *BLUEVIA 1.6*

Como se ha comentado a lo largo del texto, la plataforma de *Bluevia* ofrece varios servicios. Estos, se describen un poco más en detalle a continuación:

7.3.1 OAuth

Este API pone a disposición de las aplicaciones servicios que cubren las operaciones del proceso de autorización, para conseguir *Tokens*.

Recordemos una vez más, que la versión privada de los API no usa *Tokens* para identificar a los usuarios finales, por lo que no existirá versión privada de este API.

Ofrece dos servicios:

- Petición de credenciales temporales. De modo que un cliente pueda pedir autorización, definiendo: el modo de acceso deseado; la dirección de *callback* donde desea recibir el código de verificación, para completar el proceso; incluso los datos que cubran la transacción financiera que vaya a ser invocada más adelante con el API de *Payment.*

- Petición de credenciales finales. Este servicio no ofrece opciones en su invocación. Simplemente es autenticado con las credenciales temporales, se comprueba el código de verificación, genera unos *Tokens* finales para el usuario o la transacción, y los devuelve.

7.3.2 Advertising

Este API no necesita de la autorización de un usuario para funcionar, pués no utiliza su información personal, ni conlleva gastos para él, por lo que si es usado en solitario por una aplicación, las peticiones serán realizadas con autenticación OAuth *2-legged*.

- Expone un servicio para recuperar anuncios, ofreciendo bastantes opciones: permite elegir entre un anuncio textual, o una imagen; permite demandar anuncios relacionados con palabras específicas; permite seleccionar el grado de madurez; incluso el agente de usuario en el que se va a mostrar.

Como servicio publicitario, el uso de este API por parte del desarrollador en sus aplicaciones, conlleva un beneficio para él, cada vez que un anuncio es mostrado, o su enlace seguido por un usuario.

7.3.3 Directory

Este API ofrece a la aplicación, parte de la información que mantiene el proveedor telefónico sobre el usuario que la use.

Los sets de datos que pueden proporcionarse, comprenden: datos personales del usuario, información sobre su dispositivo móvil, información sobre el perfil de contrato, o el acceso telefónico del que dispone en ese instante.

Se ofrece desde un servicio principal, con cuatro "subservicios" que devuelven solo uno de los sets de información descritos.

- El servicio principal devuelve varios sets de datos, con todos sus campos, de una vez, dando opción a elegir dos, tres o todos.
- Mientras que si solo se necesita un set, hay que invocar el "subservicio" relativo, y se permite seleccionar los campos específicos de información relativos, que se deseen.

7.3.4 Location

Este API permite conocer la localización geográfica aproximada, en latitud, longitud y precisión estimada, en la que se encuentra un dispositivo móvil; por medio de la localización por redes de telefonía.

- El único servicio, ofrece la opción de requerir una precisión mínima en metros, y devuelve, la localización estimada.

7.3.5 MMS y SMS

Bluevia ofrece dos tipos de servicios relacionados con la mensajería telefónica. Uno enfocado al envío de mensajes, y otro enfocado a recibirlos.

El primero de ellos -Mobile Terminated, o MT- permite el envío de mensajes de texto o multimedia a través de internet, con uno o varios dispositivos móviles como destinatarios.

- Con un servicio de envío que conlleva un costo para el usuario -del que el desarrollador será en parte beneficiario-, por lo que necesita de autenticación *3-legged*. Y que ofrece la posibilidad de especificar un servidor propio, para recibir notificaciones sobre el estado de envío de los mensajes, y enviar varios archivos -hasta los 300KB- en el caso de MMS.
- Ofrece también un servicio independiente y gratuito con el que consultar directamente el estado de envío de los mensajes de forma independiente.

Por otro lado, el segundo tipo de servicios -Mobile Originated, o MO- permite la recuperación de los mensajes que lleguen a un buzón -asignado a un desarrollador, para cada aplicación-. En este caso se ofrecen servicios para:

- Recuperar la información relativa a todos los mensajes de texto existentes en el buzón: identificador de mensaje, remitente y mensaje. En el caso de MMS, se permite la opción de demandar también la información relativa a los archivos adjuntos: identificador de adjunto, nombre y tipo de archivo.
- Para el caso de MMS, se ofrece un servicio para descargar uno de los mensajes existentes al completo, junto con todos sus archivos adjuntos.
- También para el caso de MMS, se ofrece otro servicio para descargar solo un archivo adjunto en concreto de entre todos los existentes.
- Tanto para SMS, como para MMS, se ofrece además un servicio para que los mensajes entrantes, se reenvíen, o dejen de reenviarse a un servidor de la propiedad del desarrollador.

7.3.6 Payment

Esta API ofrece servicios relacionados con el pago -de diferentes cantidades, definidas previamente por *Bluevia*- de bienes, usando la factura del móvil como monedero.

Cada transacción económica que se pretenda realizar ha de ser autorizada explícitamente por el usuario final que vaya a pagarla; y una vez autorizada, se ofrecen tres servicios:

- Ejecución de la transacción. Que deberá coincidir en cuantía y descripción con la autorizada; y donde se podrá definir un servidor propio en el que recibir notificaciones del estado del pago.
- Un servicio de consulta del estado de la transacción.
- Y finalmente un servicio con el que cancelar la autorización realizada -previa ejecución de la misma-.

7.3.7 Certificación

Además de los API ofrecidos por la plataforma, *Bluevia* ofrece a los desarrolladores la posibilidad de certificar sus aplicaciones previa subida al *AppStore*.

7.3.8Compartición de beneficios

Además de los beneficios recibidos por la impresión de publicidad con el API de *Advertising; c*omo aliciente para usar algunos de los servicios de *Bluevia,* se ofrece un porcentaje de compartición de los beneficios obtenidos por el envío de la mensajería, o los pagos a partir de la factura del móvil; con el desarrollador de la aplicación -incluso cuando este no sea el productor de los bienes-. En la Figura 7-8 se muestra una tabla con los porcentajes.

CHECK WHAT'S AVAILABLE FOR EACH COUNTRY

Feature
Business model

	Argentina	Brasil	Chile	Colombia	Germany	Mexico	Spain	UK
Send SMS	✓	✓	✓	✓	✓	✓	✓	✓
Revenue share	10%		10%		10%	10%		10%
Receive SMS	✓	✓	✓	✓	✓	✓	✓	✓
Revenue share	20%		20%		20%	20%		20%
Send MMS	✓	✓	✓	✓	✓		✓	✓
Revenue share	10%		10%		10%			10%
Receive MMS	✓	✓	✓	✓	✓		✓	✓
Revenue share	20%		20%		20%			20%
Advertising	✓	✓	✓	✓	✓	✓		✓
Revenue share	50%		50%		50%	50%		50%
User context					✓		✓	✓
Location								✓
Payment	✓				✓	✓	✓	✓
Outpayment*	17%				69%	32%	46%	81%
Launch to store		✓						
Revenue share		70%						

* Merchant Revenue Share will depend on the price point selected, please check at Merchant website to get full information. The revenue share applies to net prices (without taxes).

Figura 7-8 Tabla de compartición de beneficios por país

7.4 INSTALACIÓN DEL ENTORNO DE DESARROLLO

7.4.1 *.NET* FRAMEWORK 4

Las librerías del SDK están construidas sobre funcionalidad de la versión "Cliente" del ***.NET Framework4*** de ***Microsoft***; por lo que es necesario instalarlo para poder trabajar con este acceso a *Bluevia*. Para instalar el ***Visual Studio 2010***, habrá que instalar también el *framework* completo.

1. En primer lugar, se descargará el instalador desde su página oficial[48].
2. Una vez descargado, ejecutarlo con permisos de administrador: botón derecho del ratón sobre el archivo y seleccionar "Ejecutar como administrador".
3. Aceptar los términos del contrato y seguir con el proceso. Se instalarán entonces las versiones "cliente" como la "extendida" del *framework.*

7.4.2 VISUAL STUDIO 2010

A pesar de que durante el desarrollo se han usado las versiones *Professional* y *Ultimate*, puede usarse la versión gratuita *Express* para la prueba y modificación de las librerías.

Puede descargarse esta herramienta en [49], donde seleccionaremos la versión para desarrollo en ***C#***: ***Visual C# 2010 Express***.

4. Lanzar el archivo *vcs_web.exe* con permisos de administrador: botón derecho del ratón sobre el archivo y seleccionar "Ejecutar como administrador".
5. Aceptar los términos de la licencia, y seguir con la instalación.
6. No es necesario instalar los productos opcionales que se ofrecen como se aprecia en la Figura 7-9.

Figura 7-9 **Pantalla de la selección de opciones en la instalación de *Visual C# 2010 Express***

7. Seleccionar la ubicación deseada, y comenzar la instalación.

   La herramienta comenzará a descargar los 47 MB de archivos para la instalación. Puede verse dicha pantalla en la Figura 7-10.

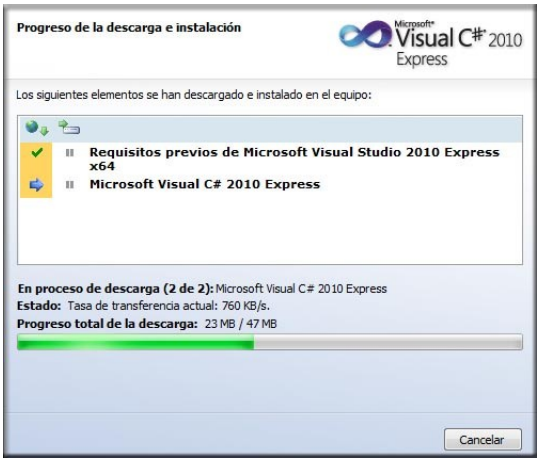


Figura 7-10 **Pantalla de descarga de los archivos de instalación de *Visual C# 2010 Express***

Tras completar la descarga, comenzará automáticamente la instalación como se ve en la pantalla de la Figura 7-11.

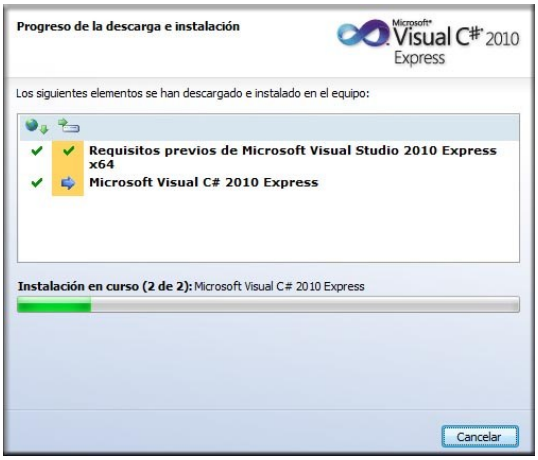


Figura 7-11 **Pantalla del proceso de instalación de *Visual C# 2010 Express***

7.5 PROXIES Y MONITORIZACIÓN DE PETICIONES

A la hora de desarrollar aplicaciones en red, un aspecto esencial es el poder conocer exactamente tanto las peticiones que se realizan a los servicios, como las respuestas recibidas de estos.

Para ello existen herramientas que funcionan como intermediarias en la comunicación -*proxies*-, que pueden monitorizar el intercambio de mensajes de red, para poder realizar una inspección al detalle.

Para el caso del desarrollo del SDK de ***.NET*** para *Bluevia* 1.6, se han utilizado dos de estos proxies: ***Whireshark***, como apoyo al desarrollo de la conectividad SSL de dos vías, para la librería de acceso privado; y ***Fiddler2*** para la inspección de los mensajes en el desarrollo de ambas librerías, pública y privada.

7.5.1WhireShark

Las comunicaciones de la librería de uso privado -*Tusted*-, se realizan a través de una conexión HTTPS, con un intercambio de certificados PEM para el SSL de dos vías como el presentado en la Figura 7-12.

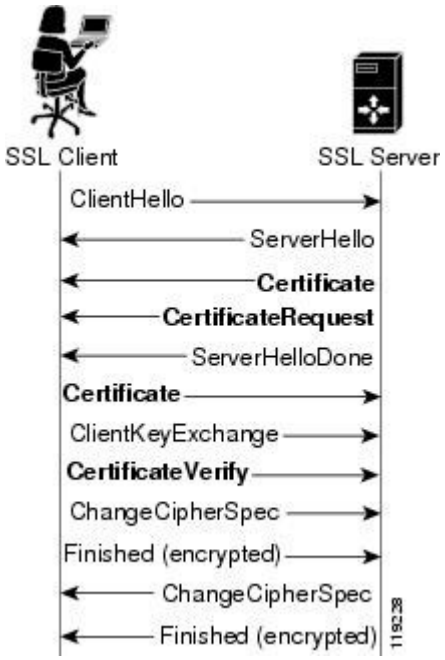


Figura 7-12 Flujo del intercambio de mensajes para una comunicación SSL de dos vías
Fuente: *CISCO* [15]

Dada la naturaleza inestable del entorno *Bluevia* de desarrollo, se hacía necesario inspeccionar en numerosas ocasiones las conexiones de los servicios privados para verificar su funcionamiento. Para ello se utilizó ***WireShark*** [19].

WireShark ofrece la posibilidad de inspeccionar la comunicación a través de toda la pila de protocolos, y comprobar al bit, todo lo transmitido y recibido; por lo que con él pueden descubrirse que certificados de cliente espera en cada momento el servidor, y en que paso de los representados en la Figura 7-12 puede fallar el proceso.

Dado que las librerías *Trusted* no forman parte del ámbito directo de este proyecto, no se profundizará más en este punto.

7.5.2 Fiddler2

Fiddler2 es un proxy útil y sencillo para realizar el filtrado e inspección de las peticiones y respuestas HTTP y HTTPS.

7.5.2.1 Inspección de peticiones con Fiddler2

Para capturar trazas de la comunicación de red entre la librería y los servicios de *Bluevia*, descifrar e inspeccionar los mensajes HTTPS; se ha usado Fiddler2.

Para que la herramienta trabaje de modo que "desencripte" los mensajes, tan solo hay abrir la ventana de opciones: "*Tools/Fidler Options…",* como se aprecia en la Figura 7-13.

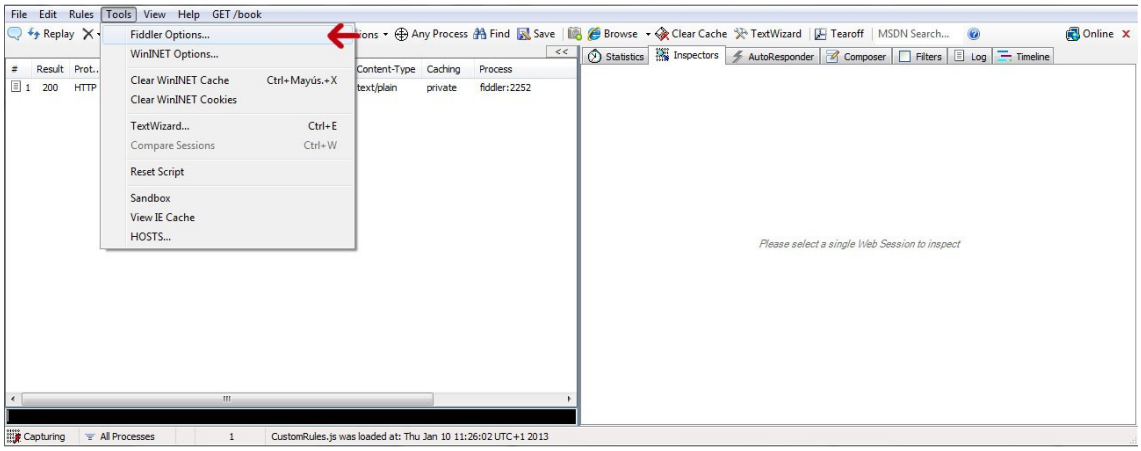


Figura 7-13 Imagen de la apertura de la configuración de opciones en Fiddler2

Y asegurarse de que están marcadas la opciones: "*Capture HTTPS CONNECTs"* y "*Decrypt HTTPS traffic*", como se muestra en la Figura 7-14.

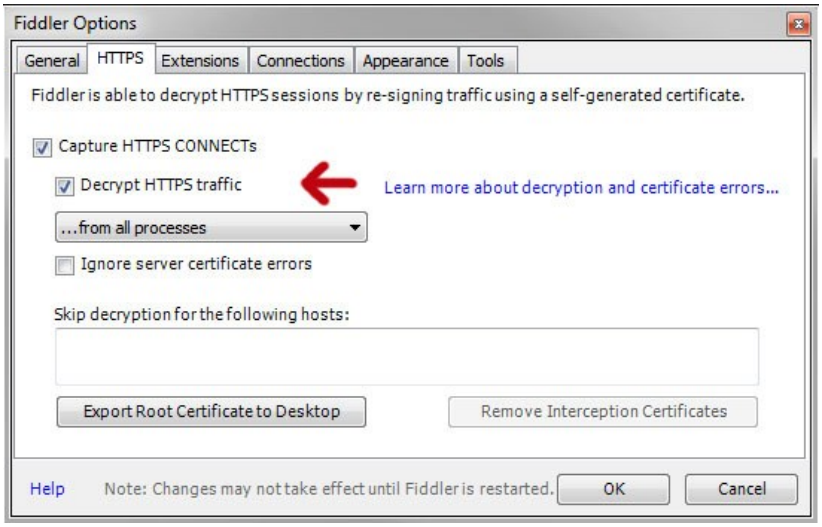


Figura 7-14 Imagen de la configuración para "desencriptar" tráfico HTTPS en Fiddler2

Una vez preparada la herramienta, en caso de que el programa no haya iniciado la captura automáticamente -en la parte inferior izquierda no aparecería el indicador de captura (2)-, se puede comenzar a monitorizar tráfico de red marcando: (1) "*File*/*Capture Traffic*", o pulsando "F12". Esto se muestra en la Figura 7-15.

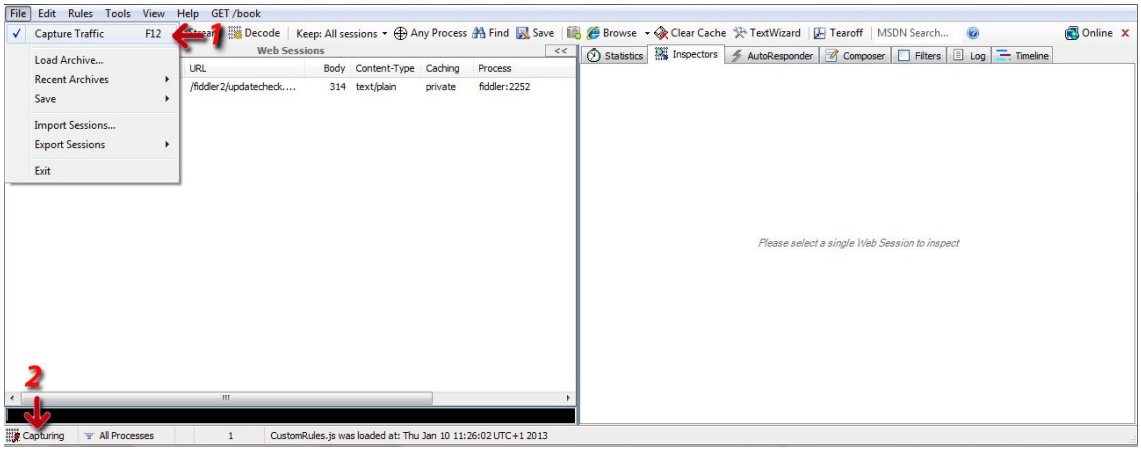


Figura 7-15 Imagen del inicio de la captura de tráfico en Fiddler2

Con el programa capturando tráfico, se puede realizar una petición para inspeccionarla en profundidad. Para ilustrarlo, se ejecuta el ejemplo de *Advertising*.

En la Figura 7-16 se aprecia como al lanzarlo, aparece una conexión con *Bluevia* (1) y una comunicación (2):

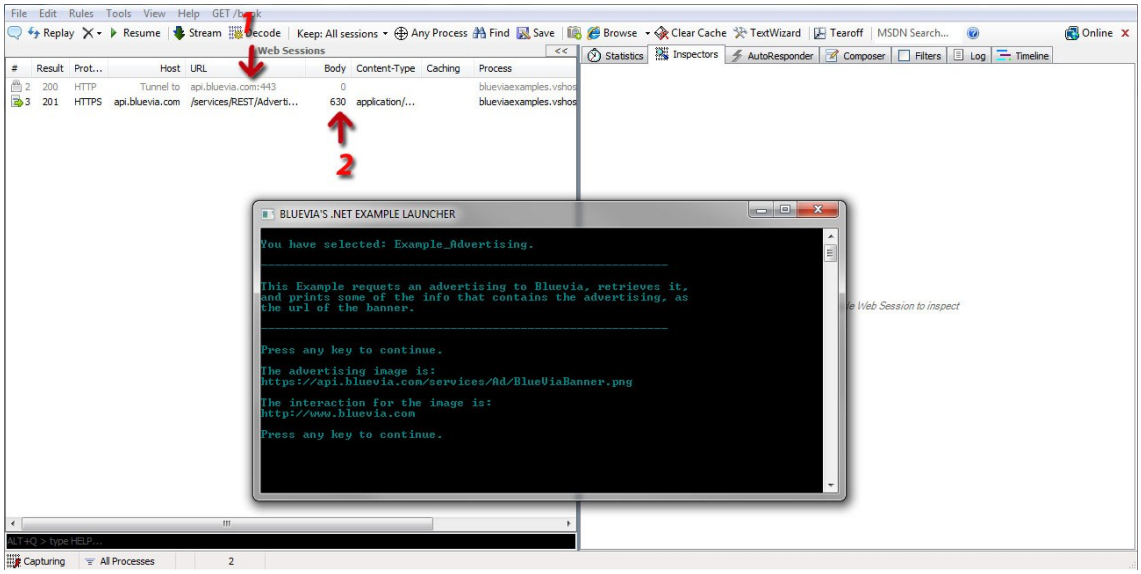


Figura 7-16 **Imagen de la captura del tráfico de red generado al ejecutar el ejemplo de *Advertising.***

Seleccionando la línea de la comunicación, se cargará toda la información relativa a esta en el campo de la derecha; y abriendo la pestaña "*Inspectors*" se mostrarán tanto la petición (1), como la respuesta (2), como muestra la Figura 7-17.

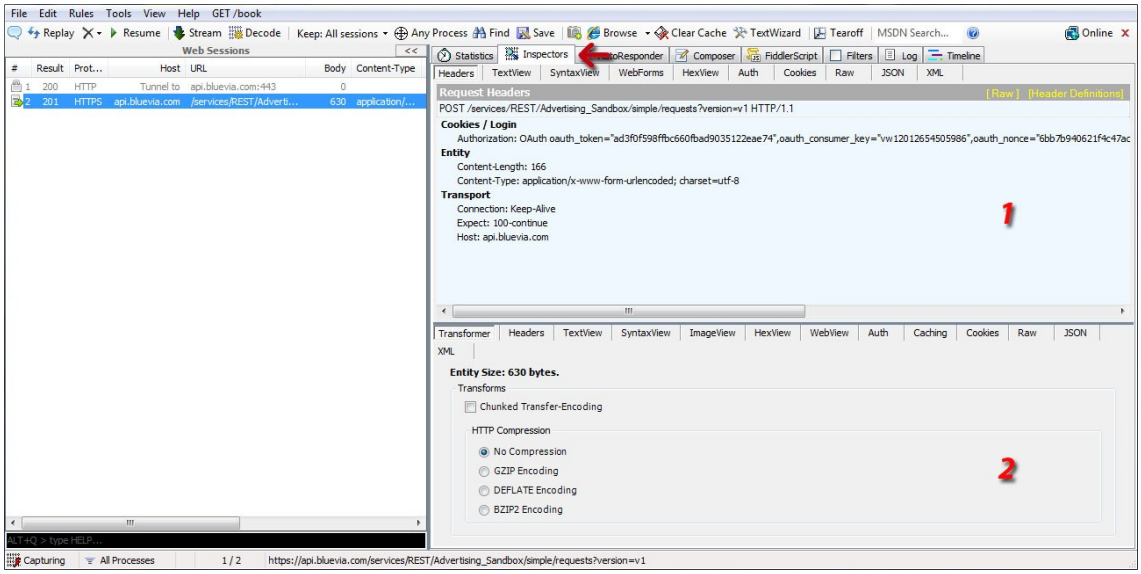


Figura 7-17 **Imagen del despliegue de la información relativa a la comunicación con el servicio de *Advertising***

Para conocer el contenido textual y completo de los mensajes -URL, parámetros, cabeceras y cuerpo-, habrá que seleccionar las vistas "*Raw*", como muestra la Figura 7-18.

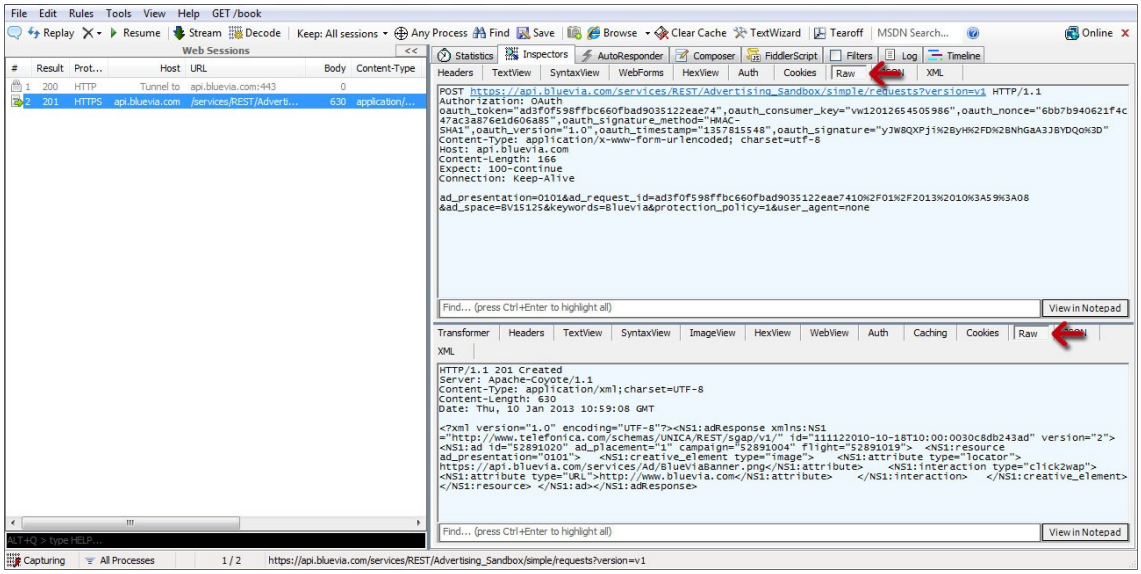


Figura 7-18 Imagen de las vistas "*Raw*" para la visualización detallada de los mensajes de red

En algunas ocasiones puede ser interesante visualizar los mensajes directamente en hexadecimal, o procesados -por ejemplo el cuerpo XML jerarquizado-; para ello basta solo con intercambiar la vista.

7.5.2.2 Inspección de peticiones *Trusted* con Fiddler2

En el caso de desear la inspección del intercambio de mensajes de los servicios de acceso privado cediendo la gestión de la conexión a ***Fiddler***, puede proporcionarse el certificado PEM de cliente al proxy.

Para ello se renombra el certificado a "*ClientCertificate.cer*", y se coloca en: *Usuario\Mis documentos\Fiddler2.*

7.6 ESTRUCTURA DEL REPOSITORIO

En el entorno de trabajo, el proyecto en desarrollo se encuentra dentro de la carpeta del repositorio "SDK git", y está estructurado de la siguiente manera para facilitar la organización del proyecto y la preparación de las versiones de entrega:

/.git: Es la carpeta generada por ***Git*** para mantener el control de las versiones.

/doc: Es la carpeta donde se guarda la documentación generada para cada versión, por lo general está vacía hasta el momento de generar una entrega. Existe en los paquetes de todos los SDK de los diferentes lenguajes.

/html: documentación -> solo existe en la entrega

/DOX: Es la carpeta donde se guardan los archivos relativos a la generación de documentación

/doxygen: Scripts y archivos de generación de la documentación para las versiones *untrusted*.

/doxygenTrt: Scripts y archivos de generación de la documentación para las versiones *trusted.*

/Magnolia: Archivo de la documentación sobre el SDK de ***.NET*** para el portal de *Bluevia*.

/lib: Carpeta donde se generan las librerías del proyecto.

/samples: Carpeta de proyectos de ejemplo. Existe en los paquetes de todos los SDK de los diferentes lenguajes.

/BlueviaExamples: Solución de ejemplos *untrusted.*

/TrustedBlueviaExamples: Solución de ejemplos trusted.

/sdk: Carpeta de Código fuente. Existe en los paquetes de todos los SDK de los diferentes lenguajes.

/libraries: Carpeta donde se guardan las librerían necesarias para la solución -proyecto-.

/Telefónica: Código fuente del sdk

/sdktest: Carpetas de proyectos para "testear" el proyecto.

/Bluevia.Tests: Tests unitarios para comprobar que no se rompe el funcionamiento de la librería con algún cambio.

/Tests: Tests de consola para apoyar el desarrollo de la librería, inspeccionar los servicios y funcionalidades de forma pormenorizada...

-LICENSE.txt: Archivo de licencia LGPL de uso de la librería. Existe en los paquetes de todos los SDK de los diferentes lenguajes.

-Readme.md: Archivo de descripción resumida para ***GITHUB***. Existe en los paquetes de todos los SDK de los diferentes lenguajes.

-Guion de entrega.txt: Archivo de descripción del proyecto para futuros desarrolladores.

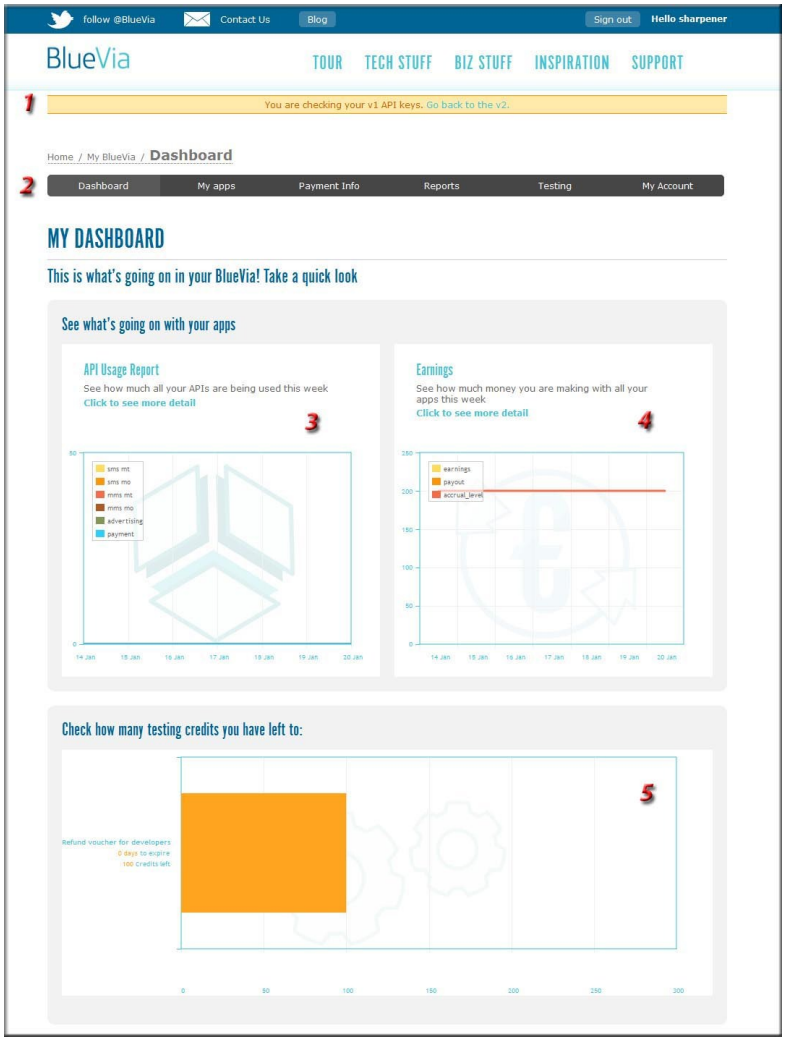
7.7 CUENTA DE DESARROLLADOR DE *BLUEVIA*, CREACIÓN DE CREDENCIALES

Para poder incluir los servicios de *Bluevia* en las aplicaciones, se ha de crear una cuenta de desarrollador. Para completar dicho trámite tan solo es necesario cumplimentar el formulario de registro en *Portal*.

Una vez creada la cuenta, puede empezar a gestionarse desde el panel del desarrollador.

7.7.1 Manejo del panel de desarrollador

En el panel de desarrollador pueden darse de alta y gestionarse las aplicaciones creadas, configurar los datos bancarios donde recibir pagos, acceder a los historiales de uso y beneficios de las aplicaciones, y controlar el uso de créditos de test. En la Figura 7-19 se muestra dicho panel.



1: A partir de 2013, las API de *Bluevia* poseen un nuevo sistema. En este campo se puede alternar entre la vista de la funcionalidad antigua -V1- y nueva -V2-.

2: Barra de secciones del panel.

3: Resumen del uso que han hecho las aplicaciones del desarrollador, de los servicios de *Bluevia*.

4: Resumen de beneficios conseguidos.

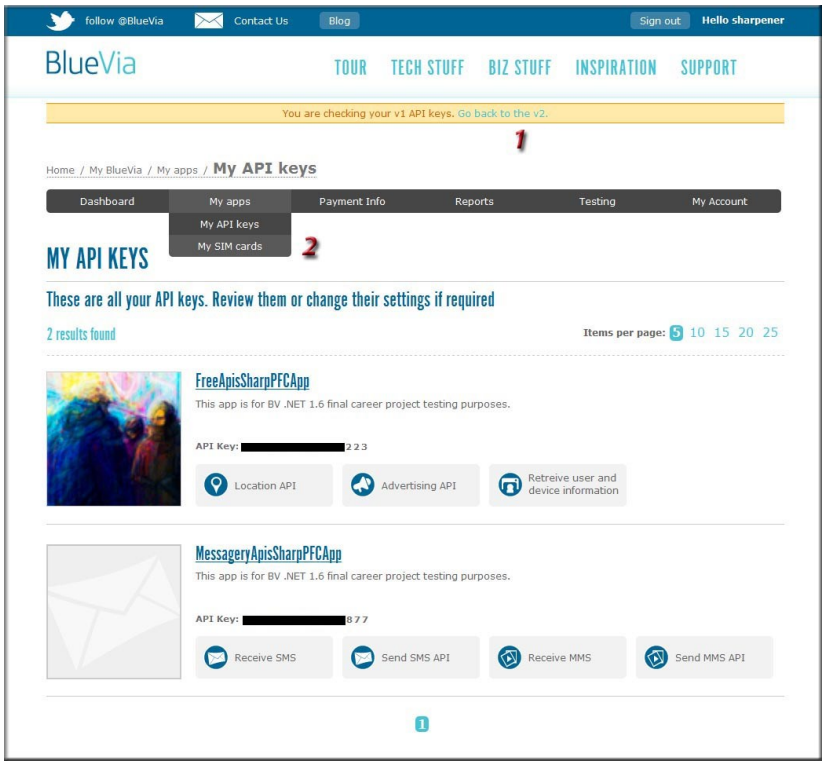
5: Créditos de test.

Figura 7-19 Vista principal del panel de desarrollador de *Bluevia*, y leyenda.

Las secciones disponibles en el panel son:

- ***My Apps***: En la que pueden gestionarse las aplicaciones existentes.

  En la Figura 7-20 presenta una imagen de la subsección *"My API Keys"*, en la que se muestra una lista de las aplicaciones registradas por el desarrollador.



1: En la imagen está habilitada la vista para las aplicaciones antiguas -V1-, por lo que no aparece la opción para crear más aplicaciones.

2: La subsección "*My SIM Cards"*, es relativa a *Bluevia* para ***Arduino***.

Figura 7-20 Imagen de la subsección "*My API Keys"*, con dos aplicaciones listadas.

- ***Payment Info***: En la que pueden gestionarse los temas económicos; cuentas en las que recibir los pagos y la información de facturación.
- ***Reports***: Donde se muestran estadísticas de uso de las API, y ganancias.
- ***Testing***: Donde se muestra la información relativa a las pruebas de las aplicaciones, con los créditos disponibles de pruebas, y las aplicaciones autorizadas para hacerlas.
- ***My Account***: Sección relativa a la información personal del desarrollador.

7.7.2 Creación de credenciales de aplicación

Para dar de alta una aplicación nueva, ha de hacerse desde la sección *"My apps/My API keys"* y pulsar el botón de "*GET A NEW API KEY mostrado en la* Figura 7-21..

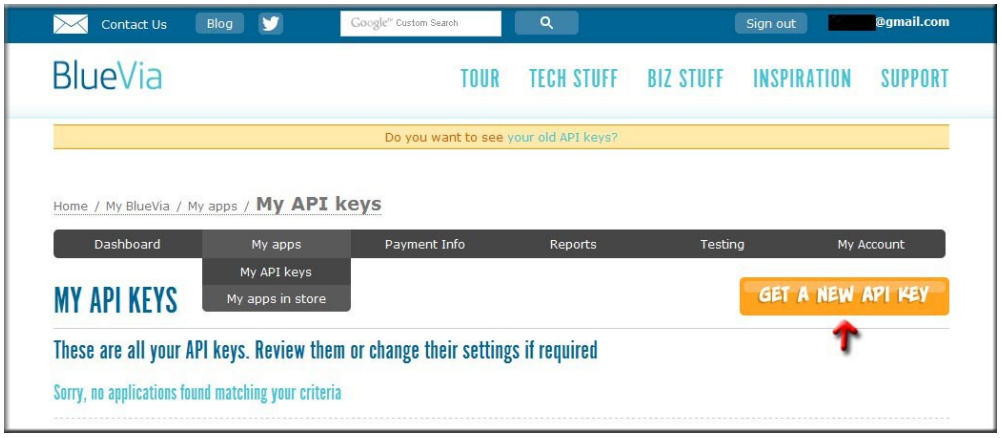
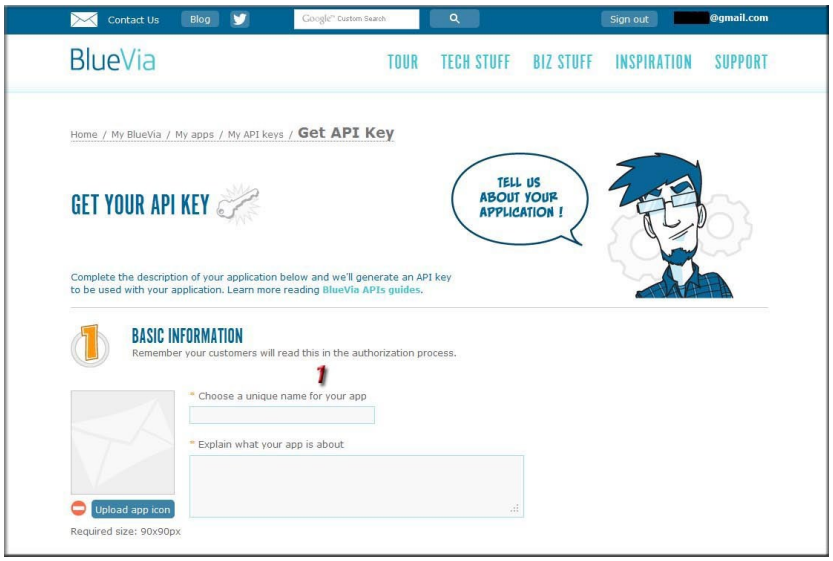


Figura 7-21 Imagen de la apertura de la sección de creación de credenciales para aplicaciones.

Tras ello aparecerá el formulario *"Get API Key"* en el que configurar las opciones de los servicios que quieran usarse para la aplicación.

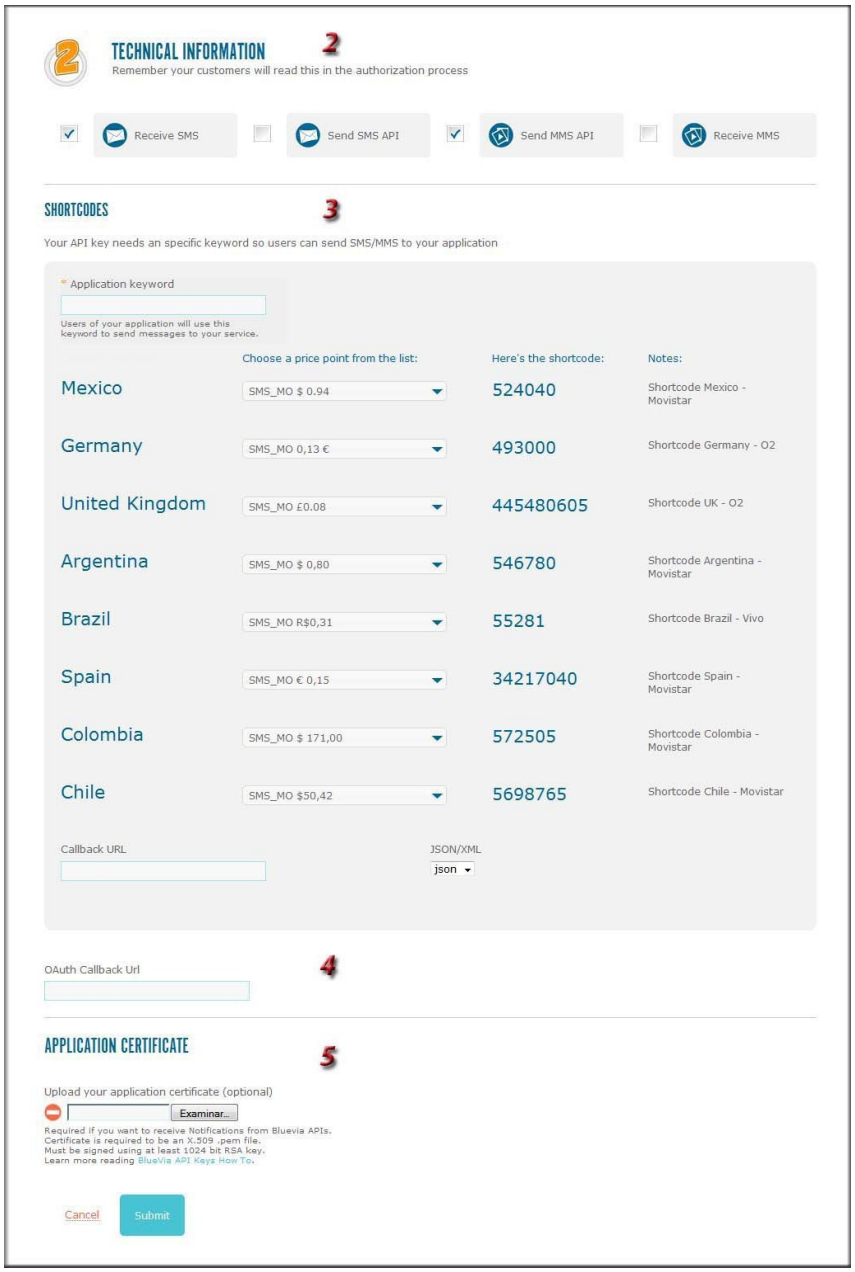
El formulario puede dividirse en dos secciones; en la primera, como se muestra en la Figura 7-22, se presentan los campos para proporcionar la información básica de la aplicación: nombre, descripción, y un icono.



1: Campo de información básica de la aplicación que vaya a ser dada de alta.

Figura 7-22 Imagen de la sección básica del formulario de alta de aplicaciones en *Bluevia*, y leyenda.

En la segunda sección, como se muestra en la Figura 7-23, pueden seleccionarse las API de Bluevia que vayan a ser usadas por la aplicación. Al seleccionar algunas de ellas, se desplegarán opciones o información extra en el formulario:



2: API disponibles para la selección. Hasta 2013 podían seleccionarse todos los API enumerados en este proyecto -V1-.

3: Información relacionada con los API de recuperación de mensajería -MO-. Campo para definir la clave para la recuperación de mensajes, precios en cada país.
Los campos inferiores son relativos a la versión 2.

4: Redirección de *OAuth*.

5: Para poder utilizar los servicios de notificaciones, hay que proporcionar el certificado PEM del servidor que vaya a recibirlas.

Figura 7-23 Imagen de la sección de selección de API del formulario de alta de aplicaciones en *Bluevia*, y leyenda.

Una vez completados los datos obligatorios, puede enviarse el formulario para terminar el proceso de registro. Finalmente se presenta una pantalla con los datos de la nueva aplicación, como la mostrada en la Figura 7-24; donde aparecen las credenciales generadas, y los datos proporcionados para el funcionamiento de esta, además de un resumen con la información del registro:

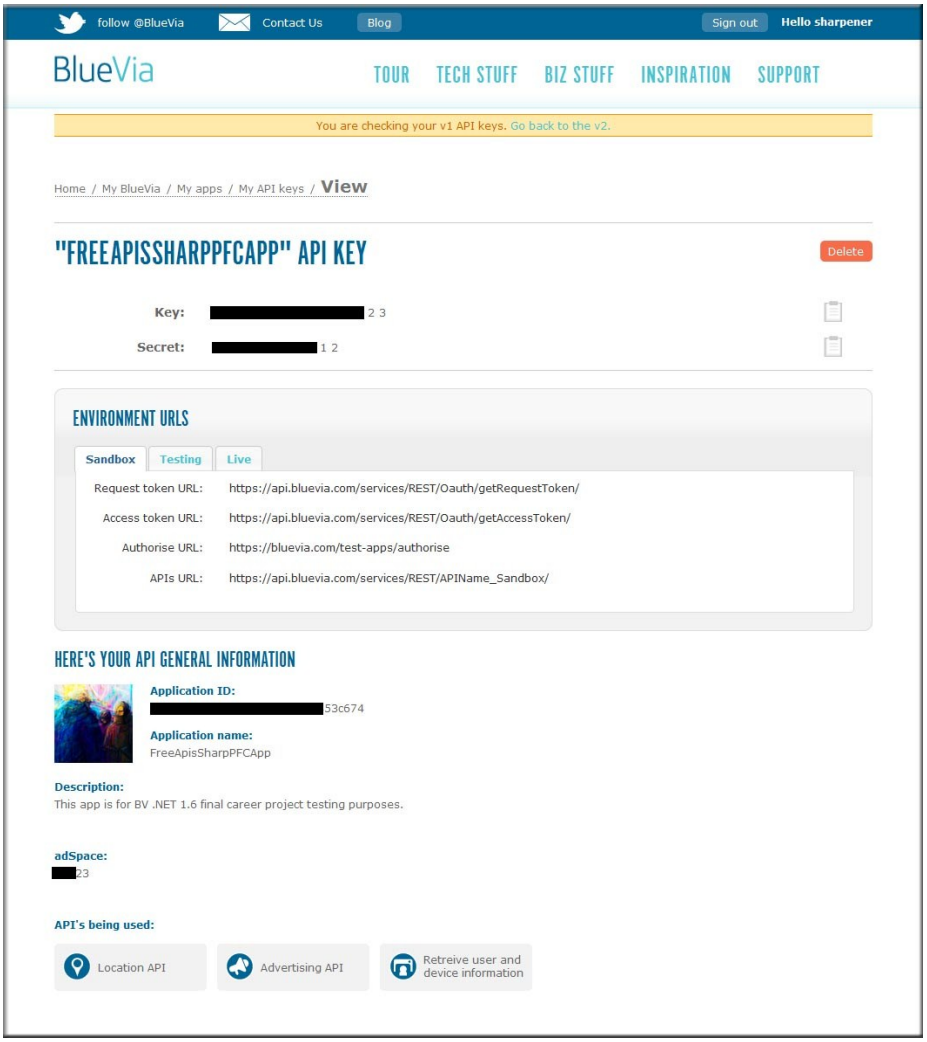


Figura 7-24 Imagen de la información referente a una aplicación del desarrollador

7.8 CREACIÓN DE CREDENCIALES PARA EL USUARIO FINAL

Para poder usar una aplicación que utilice los servicios ofrecidos por *Bluevia*, un usuario final debe de autoriazarla para conseguir unas credenciales.

Para ello la aplicación deberá realizar un proceso de *OAuth*, y el usuario aceptar las condiciones del uso de la aplicación.

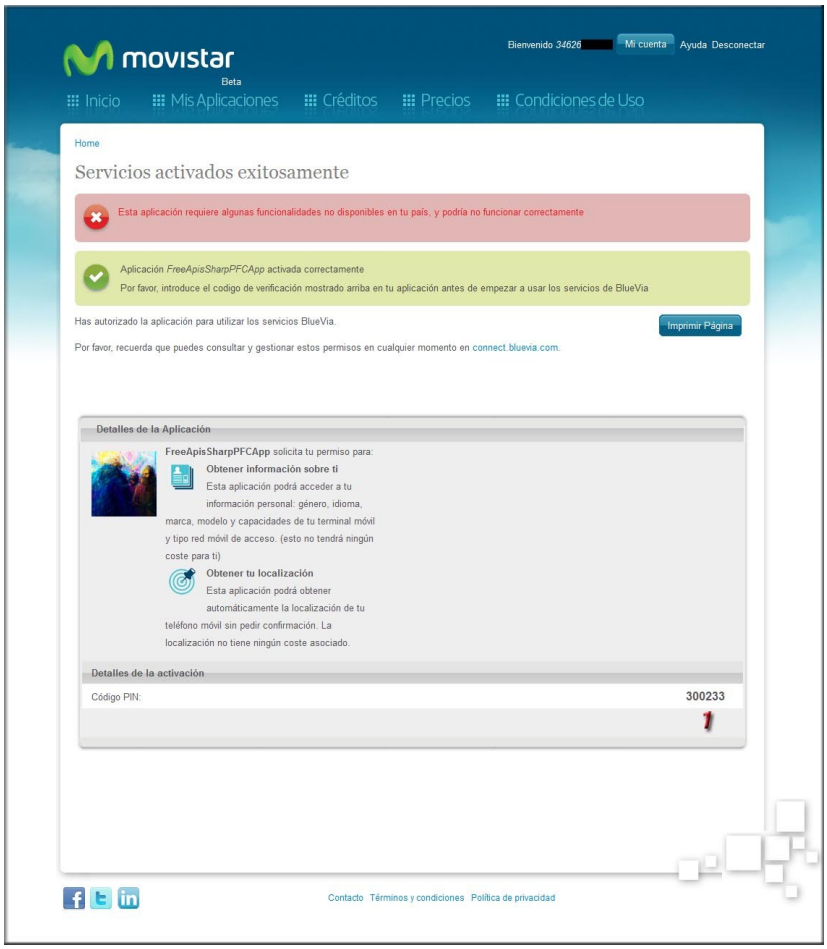
Durante el primer paso del proceso de *OAuth*, la aplicación pide unos *Tokens* temporales, y redirecciona al usuario a la página de *Connect*, con URL construida con el *Token* como parámetro: https://connect.bluevia.com/autorise?el_token

El usuario final, deberá hacer *login*, y se le presenta una página con la información de la aplicación que requiere de su autorización, para que acepte las condiciones, como la mostrada en la Figura 7-25.

1: En ocasiones la aplicación usará servicios de Bluevia no disponibles en el país del usuario final; por lo que aparecerá un aviso.

Figura 7-25 Imagen de las condiciones de la autorización de una aplicación por parte del usuario final

Si el proceso de autorización, no se completa vía SMS, o en un servidor de notificaciones propio; *Connect* proporciona un código de verificación que el usuario final debe de entregar a la aplicación, para que esta consiga los *Tokens* finales. En la Figura 7-26, se muestra dicha provisión.



1: Código de verificación para completar el proceso de *OAuth.*

Figura 7-26 Imagen de la pantalla con las condiciones aceptadas, y el código de verificación

Una vez completado el proceso, el usuario final puede ver un resumen de las aplicaciones autorizadas en la sección "*Mis Aplicaciones*" -ver Figura 7-27-.

Figura 7-27 Imagen de la pantalla de las aplicaciones autorizadas por el usuario

7.9 DOCUMENTACIÓN CON DOXYGEN

Doxygen es una herramienta de generación de documentación para proyectos de software, a través de *scripting,* que ha sido empleada para generar el manual de uso del SDK.

Ofrece una interfaz gráfica, ***Doxywizard***, con multitud de opciones para configurar el *script* de generación, y el proyecto en curso. En la Figura 7-28 se muestra una de las pantallas de opciones.

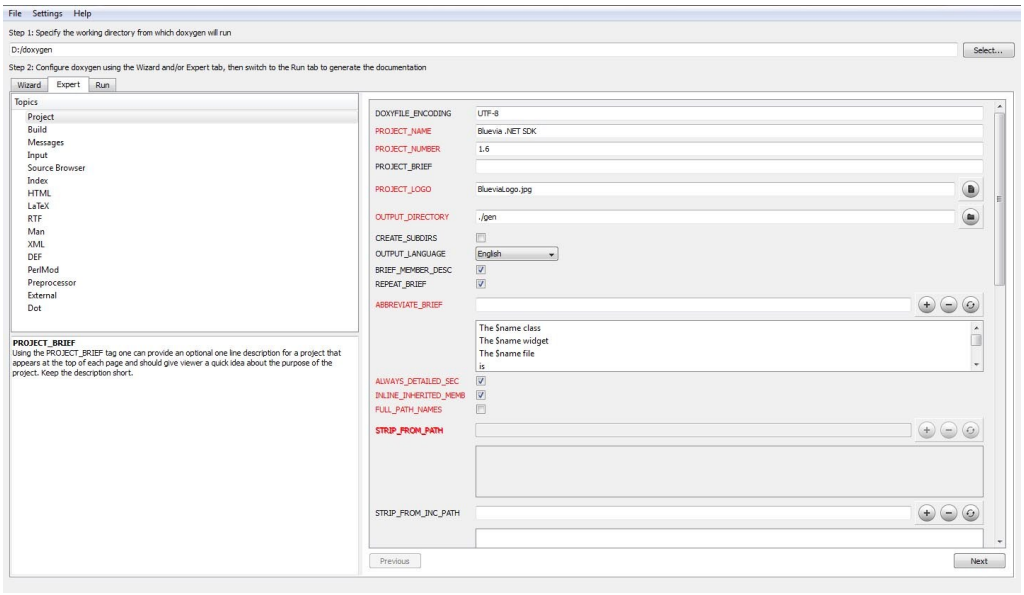


Figura 7-28 Imagen de una de las pantallas de configuración del *Doxywizard*

Para el manual de uso del SDK, además de las referencias extraídas del código fuente y sus comentarios, se han añadido guías y manuales; ***Doxygen*** también permite la inclusión de este tipo de documentación, generándola a partir de archivos con una sintaxis parecida a HTML.

Algunas de las opciones del *Script* de generación del manual son:

- Generación del manual en HTML con panel de navegación, funcionalidad de búsqueda, y diagramas de relación de clases.
- Escaneo recursivo en las carpetas del proyecto, para extraer información de todos los ficheros de la librería.
- Optimización de salida para proyectos en ***C#***.
- Extraer documentación de clases y objetos privados.

7.9.1 Un ejemplo de la documentación generada

En las figuras: Figura 7-29, Figura 7-30 y Figura 7-31, se presentan unas instantáneas de la documentación ofrecida en el paquete descargable:

7.9.1.1 Documentación de las clases

Figura 7-29 Ejemplo de las referencias de clase: BV_Connector

7.9.1.2 Guías de uso de los servicios

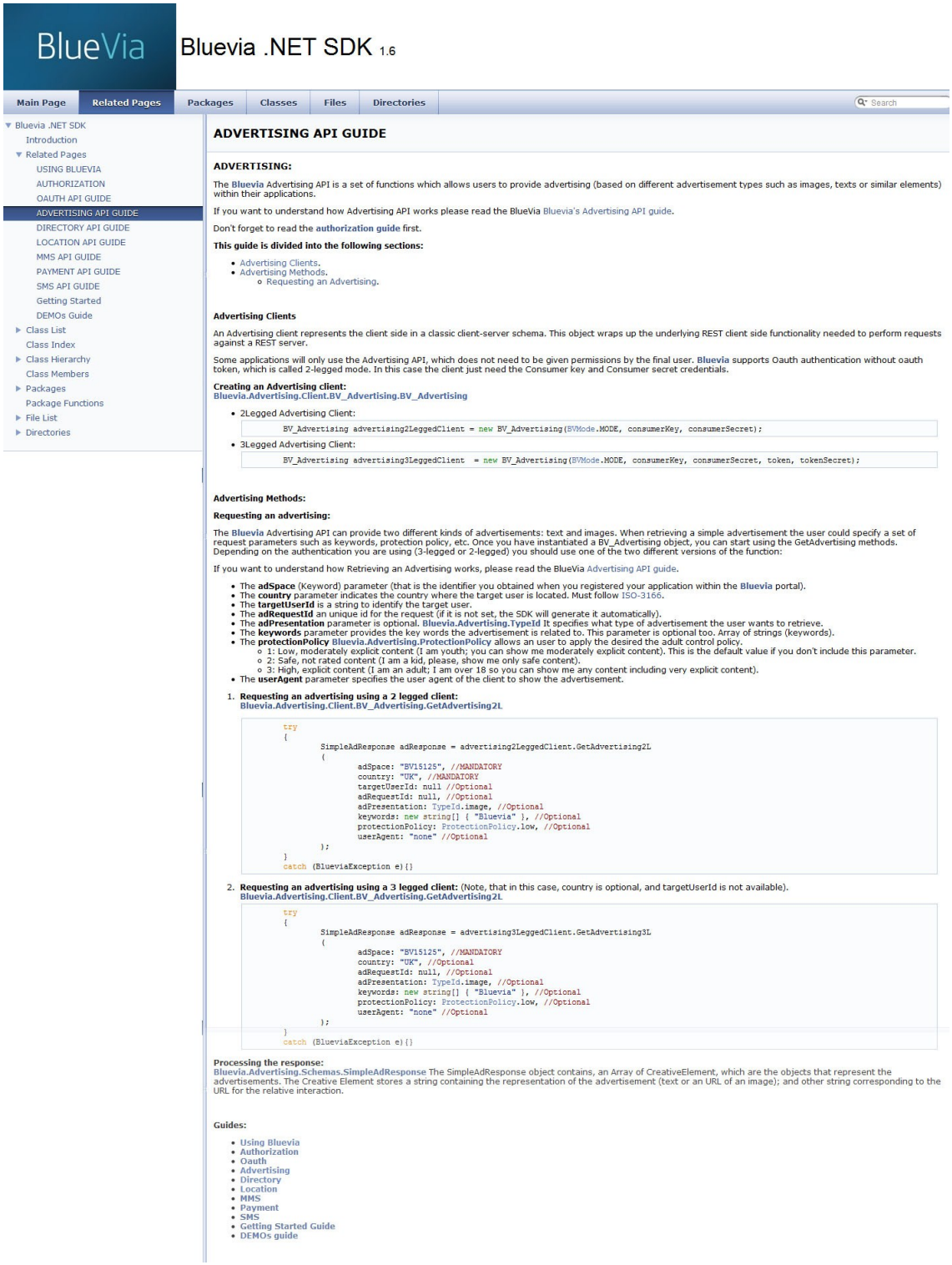


Figura 7-30 Ejemplo de las guías de uso de API: *Advertising.*

7.9.1.3 Manuales de uso del SDK

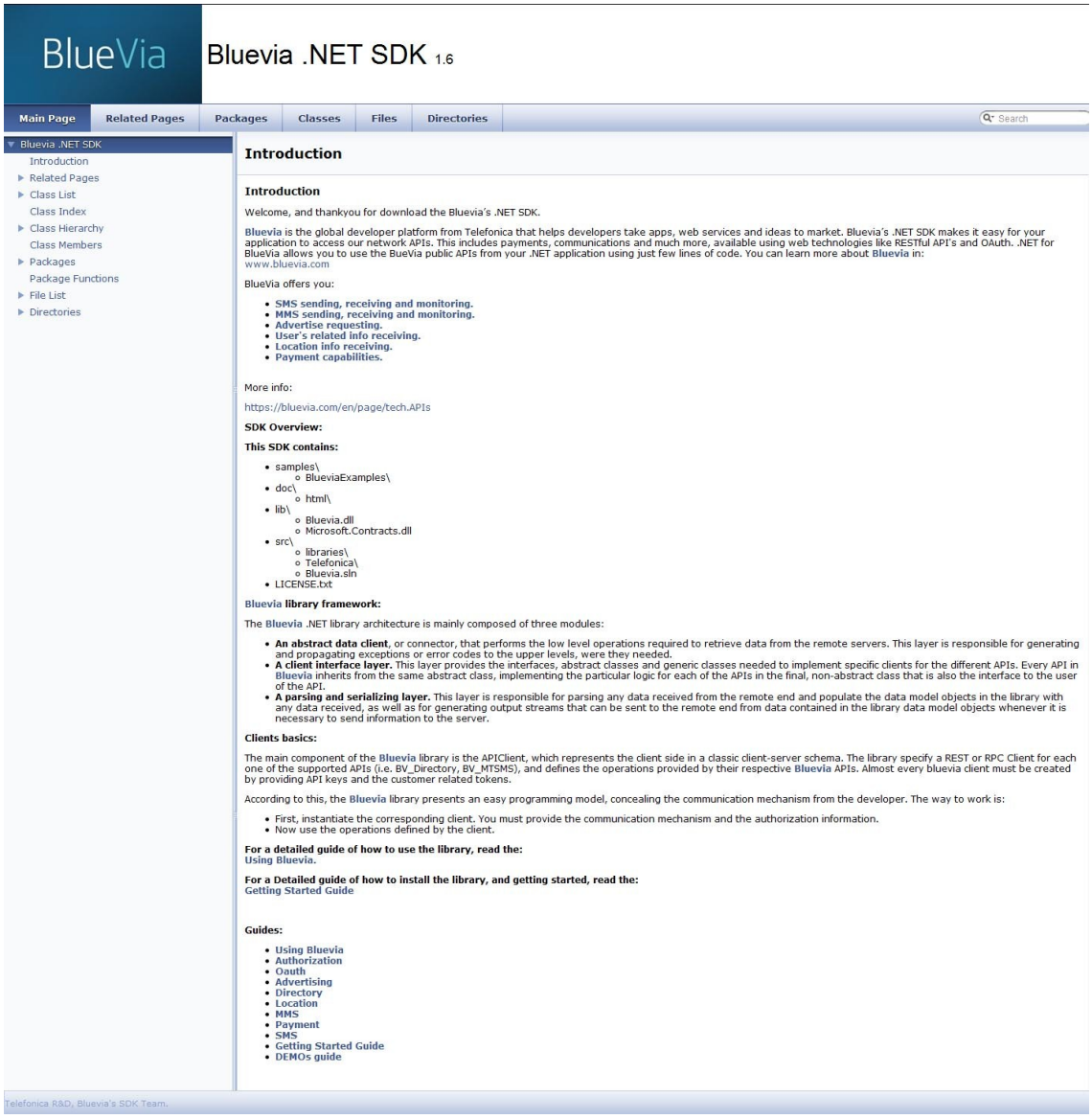


Figura 7-31 Ejemplo de los Manuales de uso del SDK: Introducción

7.10 GLOSARIO DE TÉRMINOS

2legged: Término referente a la autorización *OAuth*, para el caso en el que solo sean necesarias las credenciales de aplicación. Es usado como sistema de autorización en algunos servicios públicos de *Bluevia*, y en todos los privados.

3legged: Término referente a la autorización *OAuth*, para el caso en el que sean necesarias tanto las credenciales de aplicación, como las de usuario. Es usado como sistema de autorización en la mayoría de servicios públicos de *Bluevia*.

API: *Application Pregramming Interface*. Conjunto de rutinas que provee acceso a las funciones de un determinado software. En *Bluevia* los servicios ofrecidos se ofrecen a través de APIs web.

CRUD: *Create Retrieve Update Delete*. Conjunto de operaciones básicas que proporcionan la funcionalidad de operar con los recursos de *Bluevia.*

Diccionario: Tipo de objeto que en ***C#*** consiste en una lista de pares "Nombre-Descripción".

GNU: Proyecto de Software libre que idea la licencia GPL.

GPL: *General Public License.* Licencia de productos que permite la libre distribución y modificación de estos, a costa de licenciarlos bajo las mismas condiciones.

HTTPS: *HyperText Transfer Protocol Secured*. Implementación del protocolo HTTP enfocada a la transferencia segura de datos, ya que se monta sobre el sistema de cifrado SSL/TLS.

Inteligencia: Conjunto de lógica operativa y datos necesarios para realizar algún proceso.

JSON: *JavaScript Object Notation*. Formato de empaquetado de datos enfocado al intercambio de objetos ligero, en la web.

LGPL: *Lesser General Public License.* Licencia de librerías que permite la libre distribución y uso de estas en productos que no estén las mismas condiciones.

MIMEMultipart: Formato MIME para empaquetar varios archivos en un solo mensaje NVT ASCII, separados por patrones *boundary*.

MSISDN: Dirección de dispositivo móvil para red de telefonía.

NVT ASCII: Formarto de codificación de 7bits.

OOB: Fuera de línea. Valor del parámetro de *callback* de *OAuth,* que indica al servidor de recursos que el código de verificación sea proporcionado al usuario a través de un medio desconocido para la aplicación a autorizar.

SDK: *Software Development Kit*. Conjunto de herramientas para crear *software*.

SOAP: *Simple Object Access Protocol*. Protocolo web que define cómo pueden comunicarse diferentes procesos por medio de intercambio de datos XML.

SSL: *Secure Socket Layer.* Capa de adaptación sobre TCP que proporciona comunicaciónes seguras por la red, generalmente a servicios de la web.

TCP: *Transmision Control Protocol.* Protocolo de transmisión de datos sobre IP.

TLS: *Transport Layer Security.* Sucesor de SSL.

UID: *Unique IDentification*. Código generalmente numérico de identificación único. Utilizado en el SDK para marcar peticiones, o generar boundaries.

URL: *Uniform Resource Locator.* Sistema que especifica el formato de identificación de recursos, generalmente de Internet. Consiste en una cadena de caracteres compuesta por bloques que definene el protocolo de acceso al recurso, y su ubicación.

XML: *eXtensible Markup Language. M*etalenguaje extensible de etiquetas para definir lenguajes o estructuras para diferentes necesidades. Formato en el que se definen objetos varios objetos de intercambio en *Bluevia.*

XSD: *XML Schema Definition.* Lenguaje para describir la estructura y restricciones de los contenidos definidos por XML. En utilizado en *Bluevia* para definir los objetos de las API web, y generarlos en los SDKs de ***C#*** y ***Java*** automáticamente.

7.11 REFERENCIAS DE LA MEMORIA

Las imágenes que no han sido expresamente creadas para este documento, han sido extraídas de las siguientes Fuentes:

- El logotipo de *Bluevia*, y los iconos de los servicios, han sido extraídos de la versión 1.5 del portal de *Bluevia.*
- Las imágenes de tablas de *Bluevia*, han sido extraidas del ***Portal*** 1.6.
- El logotipo de OAuth, ha sido extraído de la página: http://hueniverse.com/oauth/
- El logotipo de ***.NET***, ha sido extraído de la página: http://blogs.msdn.com
- La imagen de las APIs Web ha sido extraída de la página: http://apievangelist.com/

[1] ***Bluevia***; Sitio principal; https://bluevia.com/

[2] ***Bluevia***; Referencia a los API disponibles en *Bluevia*; https://bluevia.com/en/page/tech.APIs

[3] ***Bluevia***; Referencia del SDK de Java en *Bluevia*; https://bluevia.com/en/page/tech.libraries.Java

[4] ***Bluevia***; Referencia del SDK de PHP en *Bluevia*; https://bluevia.com/en/page/tech.libraries.PHP

[5] ***Bluevia***; Referencia del SDK de Ruby en *Bluevia*; https://bluevia.com/en/page/tech.libraries.Ruby

[6] ***Bluevia***; Referencia del SDK de ***.NET*** en *Bluevia*; https://bluevia.com/en/page/tech.libraries***.NET***

[7] ***Bluevia***; Referencia del SDK de Android en *Bluevia*; https://bluevia.com/en/page/tech.libraries.Android

[8] ***Bluevia***; Referencia a las librerías de ***Arduino*** en *Bluevia*; https://bluevia.com/en/page/tech.arduino

[9] ***Bluevia***; Sitio en ***GitHub*** de las librerías de *Bluevia*; https://github.com/BlueVia

[10] ***Bluevia***; Especificación de servicios; https://bluevia.com/en/page/tech.APIs

[11] ***OAuth;*** RFC completa; http://tools.ietf.org/html/rfc5849

[12] ***OAuth;*** Generación de la firma; http://tools.ietf.org/html/rfc5849#section-3.4.2

[13] ***OAuth;*** Generación de la base de la firma; http://tools.ietf.org/html/rfc5849#section-3.4.1

[14] ***OAuth;*** Percent-Encoding; http://tools.ietf.org/html/rfc5849#section-3.6

[15] ***SSL;*** Guía ***CISCO*** de SSL2ways; http://www.cisco.com/en/US/docs/app_ntwk_services/data_center_app_services/css11500series/v8.10/configuration/ssl/guide/overview.html

[16] ***Windows7;*** Entrada en español de ***Wikipedia***, sobre ***Windows7***; http://es.wikipedia.org/wiki/Windows_7

[17] ***Visual Studio;*** Sitio principal; http://msdn.microsoft.com/es-es/vstudio

[18] ***Fiddler2;*** Sitio principal; http://www.fiddler2.com/

[19] ***WireShark;*** Sitio principal; http://www.wireshark.org/

[20] ***Doxygen;*** Sitio principal; http://www.doxygen.org/

[21] ***Notepad++;*** Sitio principal; http://notepad-plus-plus.org/

[22] ***Photoshop;*** Sitio principal; http://www.adobe.com/es/products/photoshop.html

[23] ***Git;*** Sitio principal; http://git-scm.com/

[24] ***Git;*** Sitio principal del repositorio ***GitHub***; https://github.com/

[25] ***Git;*** Sitio principal de la herramienta ***GITforWindows*** en ***GitHub***; http://msysgit.github.com/

[26] ***Dropbox;*** Sitio principal; https://www.dropbox.com/

[27] ***SCP;*** Entrada en español de ***Wikipedia***, sobre ***SCP***; http://es.wikipedia.org/wiki/Secure_Copy

[28] ***SCP;*** Sitio principal de ***WinSCP***; http://winscp.***NET***/

[29] ***Winrar;*** Sitio en español; http://www.winrar.es/

[30] ***MD5;*** RFC completa en español; http://www.rfc-es.org/rfc/rfc1321-es.txt

[31] ***MD5;*** Entrada en español de ***Wikipedia***, sobre ***MD5Sum***; http://es.wikipedia.org/wiki/Md5sum

[32] ***Firefox;*** Sitio en español; http://www.mozilla.org/es-ES/firefox/fx/

[33] ***Adobe Reader;*** Sitio principal; http://www.adobe.com/es/products/reader.html

[34] ***Skype;*** Sitio principal; http://www.skype.com/

[35] ***Eclipse;*** Sitio principal; http://www.eclipse.org/

[36] ***Apache Tomcat;*** Sitio principal; http://tomcat.apache.org/

[37] ***APIs Web;*** Cita en ***Wikipedia***; http://es.wikipedia.org/wiki/Web_API

[38] ***HTTP;*** RFC de la versión 1.1 del protocolo; http://www.w3.org/Protocols/rfc2616/rfc2616.html

[39] ***W3C;*** Sitio principal en español; http://www.w3c.es/

[40] ***SOAP;*** Entrada en español de ***Wikipedia***; http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol

[41] ***ProgrammableWeb;*** Sitio principal; http://www.programmableweb.com/

[42] ***Google Developers;*** Sitio enfocado a desarrolladores donde pueden encontrarse las APIs de ***Google***; https://developers.google.com/

[43] ***Amazon AWS;*** Servicios web ofrecidos por ***Amazon***; http://aws.amazon.com/es/products/

[44] ***Internet Media Types;*** Listado en ***IANA***; http://www.iana.org/assignments/media-types/application

[45] ***RPC;*** Entrada en español de ***Wikipedia***; http://es.wikipedia.org/wiki/Remote_Procedure_Call

[46] **IIS**; Página principal del producto; http://www.iis.net/

[47] ***IPhone*****;** Página principal del dispositivo; http://www.apple.com/iphone/

[48] ***.NETFramework4;*** Página de descarga del instalador; http://www.microsoft.com/es-es/download/confirmation.aspx?id=17851

[49] ***VisualStudio2010;*** Página de descarga de la versión gratuita *Express*; http://www.microsoft.com/visualstudio/esn/downloads#d-2010-express

[50] ***CLI;*** Páginas del estándar CLI: ECMA-335, ISO23271:2003; http://www.ecma-international.org/publications/standards/Ecma-335.htm http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=36769

[51] ***Proyecto Mono;*** Página principal del proyecto; http://www.mono-project.com/Main_Page

[52] ***C# Reflection;*** Guía de dicha funcionalidad en ***MSDN***; http://msdn.microsoft.com/en-US/library/ms173183%28v=vs.80%29.aspx

[53] Fátima de la Osa: "***Desarrollo de una interfaz en Google Wave para un robot conversacional de una plataforma publicitaria***"; http://e-archivo.uc3m.es/handle/10016/12614

[54] **GPL**: Página principal de la licencia: http://www.gnu.org/licenses/gpl.html

7.12 BIBLIOGRAFÍA

7.12.1 Bibliografía del SDK

- ***Msdn***: Web oficial de desarrollo para ***.NET***. Usadas guías y recursos para el desarrollo con ***C#*** y ***.NET Framework 4***; http://msdn.microsoft.com/es-es/library/ms123401.aspx
- ***Stackoverflow***: Foros web de desarrollo. Consultados para la resolución de varios problemas durante el desarrollo de las librerías; http://stackoverflow.com/
- Aex Mackey: "**Introducing .NET 4.0 With Visual Studio 2010**"; http://www.apress.com/9781430224556
- Scott Chacon, Varios autores: "***Git Community Book: The open Git resource pulled together by the whole community***"; Puede encontrarse en: http://mmc2.geofisica.unam.mx/cursos/femp/Herramientas/GIT/bookGit.pdf

7.12.1.1 Bibliografía de la memoria

- Eduardo Casilari Pérez: "***Breves notas de estilo para la redacción de proyectos de fin de carrera***"; Edición Marzo 2011; Puede encontrarse la edición de 2012 en: http://webpersonal.uma.es/~ECASILARI/Docencia/Archivos_Interes_alumno_PFC/Manual_de_estilo.pdf
- Beatriz Iglesias: "***Análisis, Diseño e Implementación de una herramienta de Gestión de Niveles de Servicio .NET integrada con Gestión de incidencias (OTRS): SLA, UC, SIP, Empresas y Usuarios***"; http://e-archivo.uc3m.es/handle/10016/10525
- Héctor Morrillas: "***Sistema distribuido de subastas dotado de calidad de servicio***"; http://www.it.uc3m.es/drequiem/

“¿Y a dónde va la recién nacida desde aquí? La Red es vasta e infinita.”

Mamoru Oshii, Ghost in the Shell